Blueprint model and language for engineering cloud applications

Nguyen, D.K.

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

DINH KHOA NGUYEN

Blueprint Model and Language for Engineering Cloud Applications

Blueprint Model and Language for Engineering Cloud Applications

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan Tilburg University op gezag van de rector magnificus, prof. dr. Ph. Eijlander, in het openbaar te verdedigen ten overstaan van een door het college voor promoties aangewezen commissie in de aula van de Universiteit op vrijdag 1 november 2013 om 10.15 uur door

DINH KHOA NGUYEN

geboren op 20 november 1983 te Ho-Chi-Minh-City (Saigon), Vietnam.

PROMOTIECOMMISSIE:

PROMOTORES:	prof. dr. Willem-Jan van den Heuvel prof. dr. ir. Mike Papazoglou
OVERIGE LEDEN:	dr. Patricia Lago dr. Claus Pahl dr. Xavier Franch

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems (SIKS Dissertation Series No. 2013-31), and CentER, the Graduate School of the Tilburg School of Economics and Management (TiSEM), Tilburg University.

To Papa, Mama, and my own little family

CONTENTS

LIST OF TABLES

List of Figures

PREFACE

Writing this page seems harder than I thought because the last five years have been such an eventful journey that cannot easily be recapped in just two pages. I came to Tilburg almost five years ago carrying a biggest dream of my life to get a PhD and now here I am, ready to turn it into reality. Looking back to the whole journey, this work would not have been possible without the help, support, and encouragement of a number of people. Below I would like to mention a few who have contributed in many ways to my success. They truly deserve my sincerest thanks.

I would like to express my first and deepest gratitude to my supervisor and promoter Prof. Willem-Jan van den Heuvel for giving me the chance to commence a PhD and for supporting me with all his best efforts to finish it. Since the very first days, I have always been impressed by his speedy working style and his ability to simplify complex problems. Under his guidance, I have had many opportunities to expand my knowledge and skills in various areas. I really appreciate that even at the last stage of my PhD he still offered his overtime in the evening to review the thesis in order to improve its quality. For all what he has done for me, I could really not imagine to have a better PhD supervisor than him.

My debt of gratitude must also go to my co-supervisor and co-promoter Prof. Mike Papazoglou for his patient and supportive guidance during the last five years. Before coming to Tilburg for a PhD, he was already known to me by his scientific reputation and that actually was the reason I decided to join this programme. I have to say it was one of the wisest decisions I have ever made in my life. Working with him has always been challenging due to his continuous demand of high-quality results. We had some really tough times discussing the thesis progress - at some points I even lost my confidence - but his steady encouragements always convinced me that it was all for my own good.

My special thanks go to dr. Patricia Lago (Vrije Universiteit Amsterdam), dr. Claus Pahl (Dublin City University), and dr. Xavier Franch (Universitat Politècnica de Catalunya), for their willingness to serve as members of my PhD committee and for their valuable feedbacks that have greatly improved my thesis.

I feel indebted to dr. Francesco Lelli due to his enormous contribution to the work presented in this thesis. Three years ago, we started working together on the very first grounding of my thesis topic. Since then, Francesco has always been a helpful daily supervisor who has guided me into the right direction. More than just a colleague, I consider him now as a friend from whom I have learned a lot. My sincere acknowledgment also goes to Alice Kloosterhuis and Mieke Smulders for their excellent support during my time in the department. They have not only helped me in administration issues but also provided valuable advices for my life in the Netherlands.

There have been many colleagues/friends in Tilburg to whom I still owe many thanks for providing me a friendly environment during my PhD life. Allow me to split the crew into two generations. Vasilios, Michael, Michele, Oktay, Willem, Cristina (the pre-marriage group): thank you for all the good (and crazy) time in Tilburg. You know what? Talking about you just reminds me of the night-outs in Kandindsky. Francesco, Yehia, Rafique, Yan, Amal, Jeewanie, Maiara, Juan, Yunwei, Sara (the post-marriage group): thank you for all the great time dining together in All-you-can-eat Sushi restaurants. I am pretty sure that no matter how apart we'll be in the future, one day we'll find ourselves together in a restaurant again.

I really had a great time with the Vietnamese community in Tilburg. Although we were just a small group, we really had some unforgettable memories (I guess the most crazy moments were my 26th and 27th birthday parties). To name a few people: Binh, Dung, Mai, anh Hai, Van, Thao, Thanh, My, Bibi, Ken, Xuan Chubby, Son, Phu, Ngoc, Giang, chi Hang, Ngoc, Hai Anh, chi Hanh, chi Hanh (2), Tuan, Phuong, Trung, Tan, Tu. Thank you all for sharing a great time with me in Tilburg.

My family has always played an important role during my PhD journey. Words are not enough to express my thanks to them for loving me, having faith in me, and standing by me throughout all the hardest times. Foremost, my beloved parents, Vu and Lien, have always had a great impact on my career life. Seeing me become a Dr. has always been their uttermost desire and I am so glad that I have finally fulfilled it. My dear sister Lien Chi deserves also a special line here as I have always been grateful to have such a wonderful sister.

Last but not least, there were two precious gifts that I have received during my PhD journey. In early 2012, Khanh Chi decided to become an integral part of my life and since then we have shared all the joyfulness and burdens. Without her sacrifice in the last period, I would not have been able to write a single line for this thesis. Our little princess, Khanh Thy, came into the world recently in June 2013. I believe that her spiritual support was the key success factor in the last stage of my PhD. Hence, this work is dedicated to her.

Dinh Khoa Nguyen
Tilburg, September 23th, 2013, 03:00 am.

CHAPTER 1

INTRODUCTION

The research presented in this thesis is positioned within the domain of engineering cloud applications. Its contribution is twofold: (1) a uniform specification language, called the *Blueprint Specification Language*, for specifying cloud services across several cloud vendors and (2) a set of associated techniques, called the *Blueprint Manipulation Techniques*, for publishing, querying, and composing cloud service specifications with aim to support the flexible design and configuration of a cloud application.

This chapter presents an overview of the thesis. Firstly, Section 1.1 introduces the *research context* in which the thesis is positioned: the intersection of the two research domains Service-oriented Architecture (SOA) and Cloud Computing. Within this context, the motivation of our research is presented in Section 1.2. Then, we identify the main *problem definition* in our motivation in Section 1.3. Section 1.4 addresses the problem definition with a concrete *research goal*, which is then decomposed into a number of distinctive research questions. In the next Section 1.5, we explain the *research methodology* that has been adopted from existing literature in design science to conduct this thesis in a systematic way. The *contributions* of the thesis are introduced in Section 1.6. We will also explain how our contributions solve the research questions. *Limitations* of our contributions are explained in Section 1.7. Finally the *structure* of the thesis is introduced in Section 1.8 that aims to provide the readers a clear reading path.

1.1 Research Context

The context of this research is at the intersection of Service-oriented Architecture (SOA) and Cloud Computing. In Section 1.1.1, we review the concept of a software service, the principles of a SOA, and how software services can be composed to build

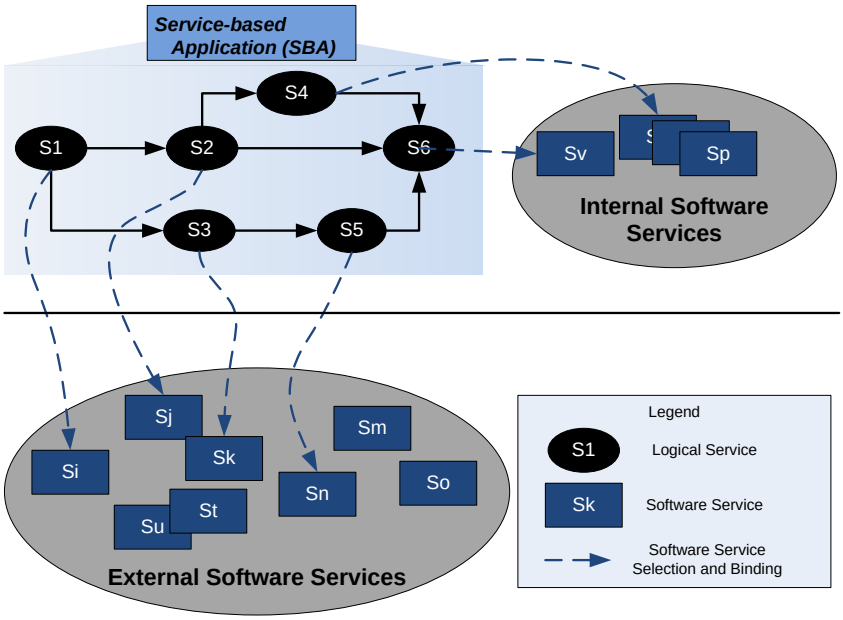
a Service-based Application (SBA) following the SOA principles. Then, Section 1.1.2 gives an overview of Cloud Computing. Finally, the intersection of SOA and Cloud Computing is explained in Section 1.1.3, which introduces the context of this thesis.

1.1.1 Service-oriented Architecture (SOA) and Service-based Applications (SBA)

Many experts see the explosive growth in **services** as the next major revolution in the world economy. 93% of the new jobs created in the U.S. between 1970 and 2000 were jobs in services [van den Heuvel, 2009]. Leading enterprises in the U.S. derived more than 50% of their revenues from services [Allmendinger & Lombreglia,]. In Europe, according to a statistical analysis of international trade in services [European Commision, 2009], the EU is the largest importer and exporter of services, followed by the USA, Japan and China. The term services used here covers interdisciplinary economic activities that create business value for an enterprise. There exist many definitions of a service in the literature, e.g., a service is "a time-perishable, intangible experience performed by a service provider for a customer acting in the role of a co-producer" [Fitzsimmons & Fitzsimmons, 2004], or a service is "an application of specialized competences (knowledge and skills) for the benefit of another entity, rather than the production of units of output [Lusch et al., 2008]. Services, on the one hand, can be understood from the business perspective as business services, which are business-oriented building blocks of an enterprise that collectively constitute key end-to-end business processes in a domain. On the other hand, services can be supported or enabled by **software services**.

Stemming from the distributed enterprise computing, **Service-Oriented Computing (SOC)** [Papazoglou, 2003] has emerged as a computing paradigm that utilizes software services as constructs to support the rapid, low-cost and easy composition of distributed applications. Building an IT system following the SOC paradigm results in a **Service-Oriented Architecture (SOA)** system, which is a logical structure of loosely coupled and interoperable software services that can be easily shared within and between enterprises, via published and discoverable interfaces [Papazoglou, 2007]. The SOA is an architecture style has many advantages over the other styles in terms of *flexibility* and *interoperability*. Software services in a SOA system are well-defined, self-contained, and context-independent software modules that can be widely reused in different contexts. The loosely coupled principle of the SOA style delivers the *flexibility* in building business applications, since these applications can be easily redesigned and reengineered whenever new business demands arise by recomposing the underlying software services. *Interoperability* in SOA can be achieved by adopting common standards for implementing SOA. The most typical way to develop a SOA system is with web service technologies that are based on the WS-* stack of standards for de-

Figure 1.1: Service-based Application composing Software Services



scribing, publishing, composing, securing, invoking, transacting and managing software services, e.g. with WSDL, SOAP, WS-BPEL, WS-Policy, etc.

Software services in a SOA system can be composed into a **Service-based application (SBA)** as defined in Definition 1.1.

Definition 1.1 (Service-based Application (SBA)) *[Andrikopoulos et al., 2008] A Service-Based Application (SBA) is composed by a number of possibly independent services, available in a network, which perform the desired functionalities of the architecture. Such services could be provided by third parties, not necessarily by the owner of the service-based application. Note that a service-based application shows a profound difference with respect to a component-based application: while the owner of the component-based application also owns and controls its components, the owner of a service-based application does not own, in general, the component services, nor it can control their execution.*

Typically, an SBA is designed to support an end-to-end business process. Figure 1.1illustrates the idea of composing software services into a process-based SBA to support (parts of) an end-to-end business process. Each step of the SBA is designed as a *Logical Service* that captures the required functionality and non-functional properties (such as QoS properties). A logical service can be realized by an internal or external *Software Service*. Internal software services are typically implemented in-house, hosted on-premise, and can be shared across different units. In contrast, the idea of reusing external, third-party software services aims to incorporate a large variety of non-competent functionalities in an SBA. The SOA principles promote the flexible

(re-)design of SBAs as it allows SBA engineers, depending on their current business demands, to pick and choose appropriate software services to implement the logical services. To be noticed, an SBA, as depicted in Figure 1.1, can be considered as a composite software service that may be reused in another SBA composition. Hence, the two terms "SBA" and "composite software service" are sometimes used interchangeably.

Despite all the advantages of building SBAs following the SOA principles, a serious limitation is that SOA development does not make any assumptions regarding the deployment and provisioning of the constituting software services of an SBA. An SBA engineer typically leaves it up to the discretion of the developers (in case of an in-house software service) and external providers (in case of reusing an external software service) to choose the deployment environment. Hence, constituting software services of an SBA are usually delivered as monolithic, "one-size-fits-all" blocks, since once the software service is deployed, it is bound to a proprietary platform and infrastructure and thus difficult to be customized and extended. Furthermore, SOA development tends to follow a "big design upfront" philosophy where it is believed that all the consumers' requirements, e.g. regarding the performance and resource consumption, have been gathered prior to implementing and deploying a software service. This leads to the situation where resources for provisioning a software service are usually consumed more than actually needed, yet the software services still cannot cope with unexpected performance loads during peak periods, e.g. when an order processing service is overloaded during busy shopping periods or when a telecom service has to accommodate an extremely large amount of SMS communication at the new year's eve.

To address this serious shortcoming, enterprises are progressively adopting cloud computing[1], which aims to provide an economy of scale of a shared infrastructure resource as well as a flexible pay-per-use service delivery and deployment models. In the next Section 1.1.2 we will discuss the topic of cloud computing and the service delivery models in the cloud.

1.1.2 Cloud Computing

Recently, the field of **Cloud Computing**, where computational, infrastructure and data resources are available on-demand from a remote source, has become an emerging research topic in response to the shift from product-oriented economy to service-oriented economy and the move from focusing on software/system development to addressing business-IT alignment. One of the reasons for its popularity is because cloud computing gives the option to outsource the operation and maintenance of IT

[1]Gartner Inc (`http://www.gartner.com`), a world's leading IT research and advisory company, has predicted that Cloud Computing will be an integral part of an IT system in 2013

tasks, allowing organizations and their employees to concentrate on their core competencies. This, together with pay-as-you-go billing that reduces the need for capital expenditure on equipment, means that with cloud computing, software services can be easily designed and tailored to a variety of business's individual requirements.

The US National Institute of Standards and Technology (NIST) defines cloud computing as "a consumption and on-demand delivery computing paradigm that enables convenient network access to a shared pool of configurable and often virtualized computing resources (e.g., networks, servers, storage, middleware and applications as services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [Mell & Grance, 2009] . From this definition, they also explain the five key characteristics of cloud computing:

1. On-demand self-service: virtualized cloud computing resources are delivered as on-demand services and users can access and manage the resources themselves.

2. Ubiquitous Network Access: Cloud computing resources should be accessible from various locations.

3. Location-independent Resource Pooling: Resources in cloud computing, provided by various vendors, should be gathered into a shared, virtualized pool. Users from any location should be able to access these resources.

4. Rapid Elasticity and Provisioning: Cloud Computing resources are elastic and thus can be rapidly provisioned to the users according to their demands.

5. Pay-per-used measured services: Cloud Computing resources are provided as services to the users and the users only have to pay for what they have used.

Figure 1.2 presents the four deployment models and three service models in cloud computing defined by NIST [Mell & Grance, 2009]. The following four deployment models reflect the four different scenarios of how the cloud is deployed, used and managed [Mell & Grance, 2009] [Armbrust et al., 2009] [Papazoglou, 2012]:

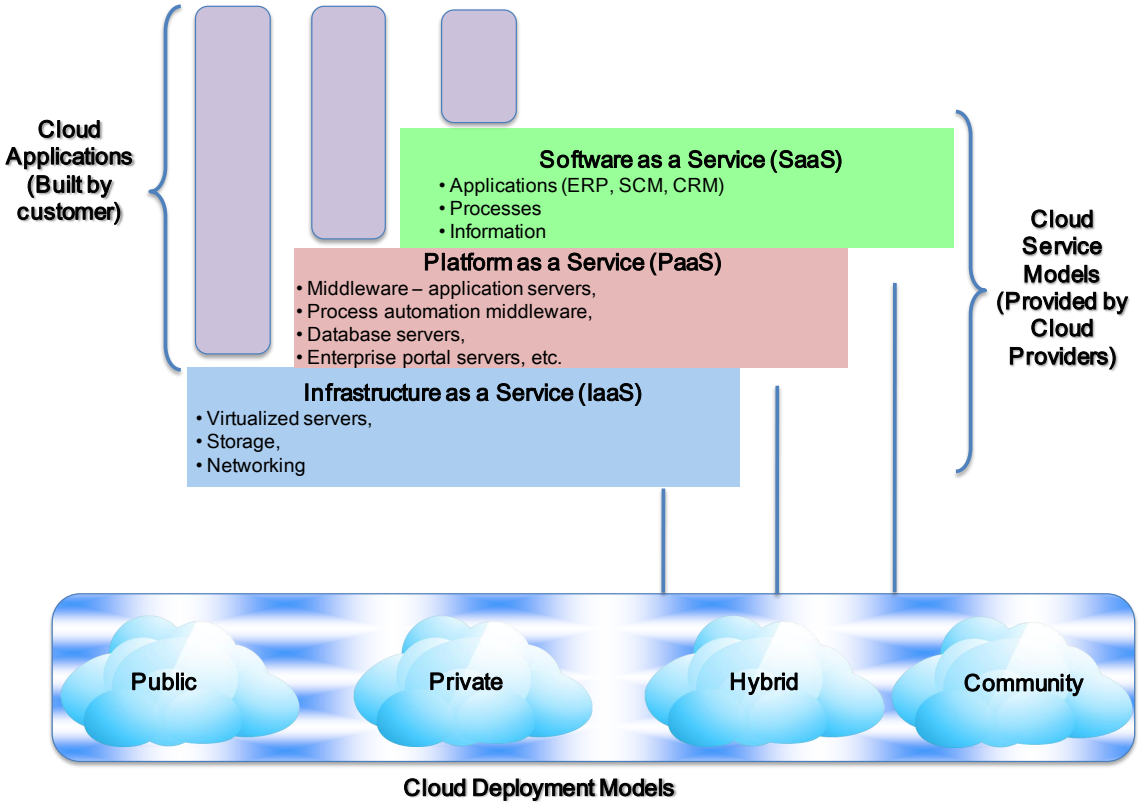
- Private cloud: The cloud solutions are developed and provisioned "on-premise" for exclusive use by a single organization.

- Community cloud. The cloud solutions are developed and provisioned for exclusive use by a specific community of organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations).

- Public cloud. The cloud solutions are developed and provisioned for open use by the general public.

- Hybrid cloud. The cloud solutions are integrated and federated solutions across both public and private clouds.

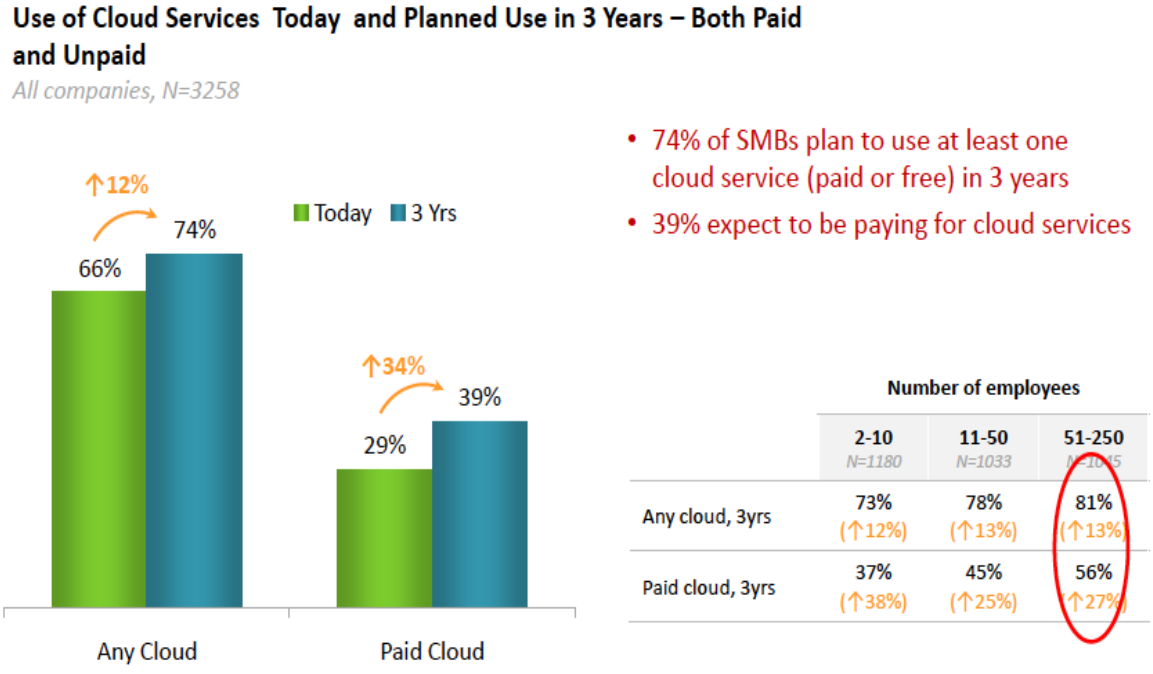
Figure 1.2: Service Models and Deployment Models in Cloud Computing



Furthermore, cloud computing is typically divided into three models of delivering services: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), as shown in Figure 1.2. These three models are usually referred to as the three cloud service layers of the cloud computing stack. Cloud services on each layer not only encapsulate the virtualisation, management, and provisioning of resources on that layer, but also define the development models for building cloud applications. In the following, each cloud service layer is explained [Taher et al., 2011][Mell & Grance, 2009] [Armbrust et al., 2009] [Papazoglou, 2012]:

- Infrastructure as a Service (IaaS): is the delivery of hardware (server, storage and network), and associated software (operating systems virtualisation technology, file system), as a service. It is an evolution of traditional hosting that does not require any long-term commitment and allows users to consume resources on-demand. IaaS incorporates the capability to abstract resources as well as deliver physical and logical connectivity to those resources and provides a set of APIs, which allow interaction with the infrastructure by consumers. Amazon Web Services Elastic Compute Cloud (EC2) and Secure Storage Service (S3) are examples of IaaS offerings. Rackspace's Mosso, GoGrid's ServePath, and flexiscale are other sample IaaS offerings

Figure 1.3: SMB Cloud Adoption Study Dec 2010 -Global Report, Edge Strategies, March 2011



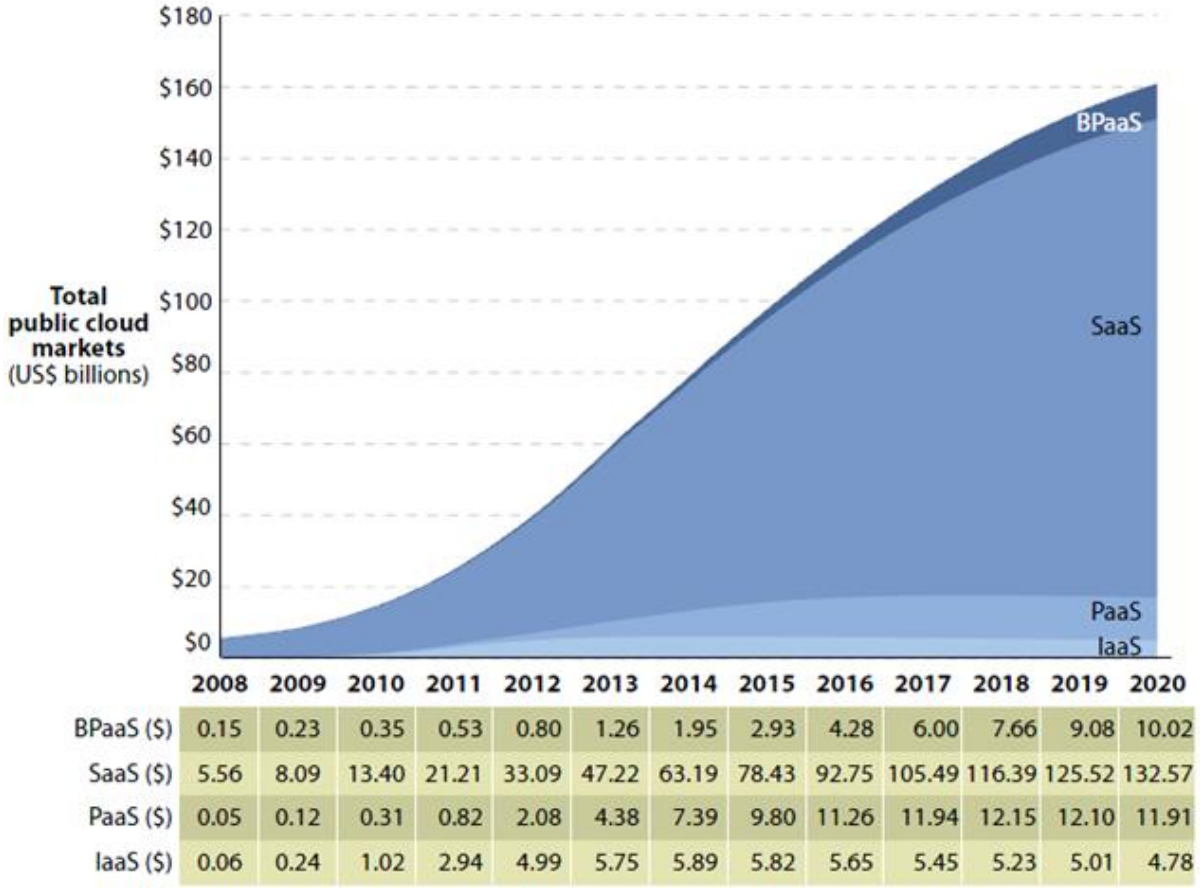
Source: SMB Cloud Adoption Study Dec 2010 -Global Report, Edge Strategies, March 2011

- Platform as a Service (PaaS): is an application development and deployment platform delivered as a service to developers over the Web. PaaS facilitates development and deployment of applications for the application developers. A PaaS offer comprises infrastructure software, and typically includes a database, middleware and development tools for delivering Web applications and services through the Internet. Consumers can create applications by, for instance, making use of commercial PaaS solutions such as the Salesforce's Force.com, Google's App Engine, or Microsoft's Azure. The consumer's application, however, usually cannot access the infrastructure enabling the PaaS.

- Software-as-a-Service (SaaS): is an "on-demand" application delivery model over the Internet built upon the PaaS and IaaS stacks. Upon user request, a SaaS provides the entire software application including its content, presentation, application logic, and management capabilities. The SaaS consumers can only access the exposed functions of the application. A typical example is SalesForce that offers CRM applications accessible by subscription over the Web. However, contemporary SaaS solutions provide only integrated functionalities built directly into the offering with no option for consumers to extend or modify them.

Cloud computing is transforming the way applications are built and provisioned to

Figure 1.4: Forecast: Global Cloud Public Market Size, 2011 to 2020 - Source: [Forrester Research Inc, 2011]



the end-users. Nowadays, a number of cloud services provided in the market has introduced new possibilities for outsourcing the development, hosting and maintenance of applications. For instance, apart from the option of being developed and hosted on-premise, a Customer Relationship Management (CRM) application can reuse some CRM functionalities provided as SaaSs by SalesForce, or it can be developed and deployed on pre-configured PaaS solutions such as Google App Engine or Microsoft Azures. In a study of cloud service adoption in 2010 [Kazarian & Hanlon, 2010], 74% of the Small-and-Medium Businesses (SMBs) were planning to use at least an external cloud service within the next three years, i.e. until 2013, and 39% are willing to pay for cloud services. Figure 1.3 presents the result of this study by Edge Strategies. Another interesting issue in this study is that for both unpaid and commercial cloud services, the number of SMBs that are willing to adopt cloud services will increase a lot in 2013.

Another interesting market research for cloud service adoption has been conducted by Forrester Research in [Forrester Research Inc, 2011]. Figure 1.4 presents the outcome of this research that indicates the global public market size of each type of cloud

services. Looking at the result, we see that from the years 2012-2013 onwards, the market size of PaaS and IaaS solutions will remain stable, whilst the market size of SaaS solutions will increase dramatically. The stability of PaaS/IaaS market can be explained by the dominant role of big players like Amazon, Google, VMWare, GoGrid, etc. Nowadays, PaaS/IaaS consumers always think of these vendors when they need PaaS/IaaS solutions for deploying and provisioning their applications.

Given the stability of the PaaS/IaaS market size in the next 5 to 7 years, SaaS providers do not have to worry about the hosting and maintenance of their SaaS applications, and thus can concentrate on their core competence of developing good SaaS functionalities. By taking into account the SOA principles of reusing and composing SaaSs, existing SaaS providers will continuously expand their solution space whilst newcomers will always discover good opportunities in the SaaS business. Hence, it is understandable that the SaaS market size is predicted to expand dramatically.

It is interesting to see the appearance of a new concept "Business Process as a service (BPaaS)" as another cloud service delivery model available on the cloud market from 2014. The term BPaaS is used for a cloud ecosystem putting focus on its enterprise-specific services. A BPaaS offers a unique end-to-end business process that is usually syndicated with other external services (possibly provided by diverse SaaS providers in the cloud) [Papazoglou & van den Heuvel, 2011].

1.1.3 The Intersection of SOA and Cloud Computing

Whilst SOA is an architectural style for building SBAs based on the reuse and composition of loosely coupled software services [Papazoglou, 2007], "the cloud serves as a good way to deploy services in an SOA environment" [Krill, 2009]. It has been agreed among several practitioners that although SOA and Cloud are highly complementary, there is one fundamental difference in their definitions: "Cloud computing is a deployment architecture, not an architectural approach for how [to] architect your enterprise IT [as SOA is]" [Krill, 2009].

Cloud services are the logical extension of software services in a SOA system in that they do not only cater for selectively composing SBAs using discrete SaaSs, but also enable the SBA engineer to find appropriate underlying PaaSs and/or IaaSs as the deployment environments for the SaaSs. In the era of SOA without the cloud, the reuse and composition of software services for a SOA system typically remained within an organization's boundary, and the software services were typically deployed and managed on an in-house proprietary platform and infrastructure.

With the stability of the PaaS/IaaS market and the dramatic expansion of the SaaS market, Figure 1.4 shows that in the near future an SBA engineer will be able to leverage the growth of the cloud service market in general and the SaaS market in particular to build a tailored SBA solution. The intersection of SOA (as an IT architecture style)

and Cloud Computing (as a deployment architecture) signifies a novel approach for engineering SBAs that supports the reuse and composition of heterogeneous cloud services offered by various cloud vendors. Following this approach, an SBA engineer is able to pick and choose freely among a pool of cloud services, irrespective of their delivery models. For instance, an SBA engineer may want to integrate a new SaaS into his SBA or to find an appropriate PaaS/IaaS to deploy a SaaS. He will be able to take advantage of the different functionalities and non-functional properties (e.g. performance, reliability and cost) of alternative cloud services to build a tailored SBA solution that meets his specific business requirements.

SBAs that are developed by reusing and composing cloud services across all three layers of the cloud stack are called Cloud Service-based Applications. Definition 1.2 gives the formal definition of an CSBA. The research in this thesis is positioned in the domain of engineering CSBAs.

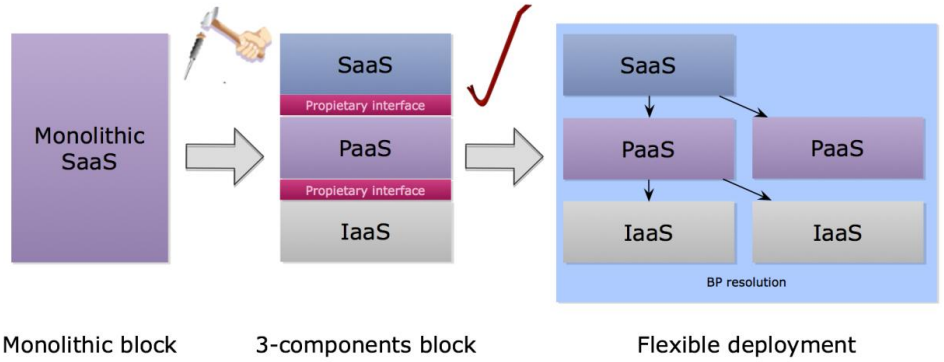
Definition 1.2 (Cloud Service-based Application (CSBA)) *A Cloud Service-based Application is a type of SBA (see Definition 1.1) that is developed by reusing and composing cloud services across all three layers of the cloud stack, i.e. SaaS, PaaS, and IaaS.*

1.2 Motivation

Engineering an CSBA comprises of the tasks of selecting and composing discrete SaaSs as well as choosing appropriate PaaSs/IaaSs for their deployment environments. However, one of the limitations of the contemporary SaaS solutions in the market is that they are usually delivered as *monolithic, one-size-fits-all* solution stacks that are predominantly tethered to the proprietary platforms and infrastructure of a cloud vendor. The customers of a monolithic SaaS usually cannot access its enabling platform/infrastructure to make some changes in its deployment environment. As a result, monolithic SaaS offerings are more likely to show failure in meeting the requirements of different consumers and usually lead to a vendor lock-in situation. The vendor lock-in situation is understood as the tremendous efforts required to customize, extend and integrate the SaaSs across different providers. For instance, existing SaaSs from providers like SalesForce[2], Google[3] or SaaSDirectory[4] are virtually not customizable, extendable or interoperable. There have been some initiatives in bringing SaaS offerings from different providers into a joint solution, e.g., the Appirio CloudFactor[5] combines the SalesForce's CRM SaaS offerings with the Google Apps' SaaS offerings.

[2]SalesForce CRM: http://www.salesforce.com
[3]Google App for Business: http://www.google.com/apps/intl/en/business/index.html
[4]HRM SaaS Applications: http://www.saasdir.com/category/humanResources.aspx
[5]Appirio CloudFactor: http://www.cloudfactorapp.com/

Figure 1.5: Breaking the monolithic SaaS solution stack



However, we observe the lack of a generic vendor-indepenedent approach for integrating SaaS offerings across multiple providers.

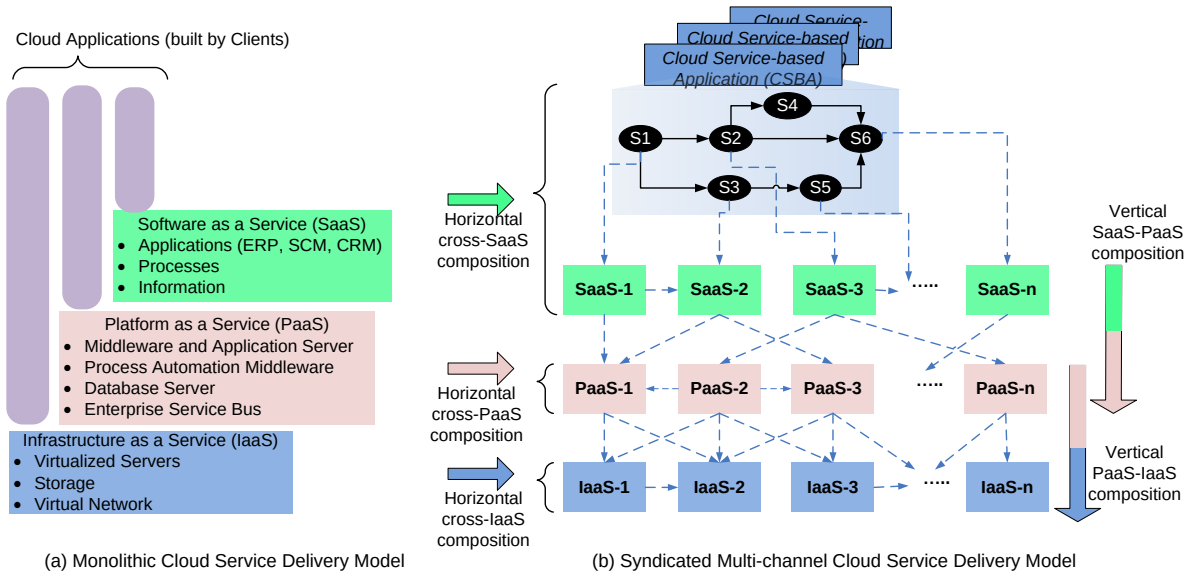
Breaking down the monolithic SaaS offerings

Engineering CSBAs using only monolithic SaaSs has the limitation that the CSBA engineers are locked into a predefined set of SaaS functionalities with little or no room for customization and extension. We see a need to break down the monolithic SaaS offerings into cloud services across three layers of the cloud stack to enable the platform- and infrastructure-agnostic engineering of CSBAs. This approach will allow CSBA engineers to freely use a variety of cloud services on all three layers, i.e. SaaS, PaaS, and IaaS, to build a tailored CSBA solution.

The idea of breaking down the monolithic SaaS offerings is illustrated in Figure 1.5. It aims to provide a more flexible method for CSBA engineers to select, customize and integrate cloud services. This approach allows for the tailoring of CSBAs to specific business needs using a mixture of alternative SaaS, PaaSs and IaaSs. On the one hand, CSBA engineers can easily couple their CBSAs in whole or part with external SaaSs. On the other hand, they also have the flexible choice of platform and infrastructure options to deploy the SaaSs.

Figure 1.6(b) introduces an approach proposed in [Papazoglou & van den Heuvel, 2011] that leverages the idea of breaking down the monolithic SaaS stacks and subsequently proposes a "syndicated multi-channel cloud service delivery model" for engineering CSBAs. This approach is compared with the current cloud application engineering approaches using the monolithic cloud services in Figure 1.6(a). It is shown that current monolithic cloud services permeate the cloud today by enforcing one-way vertical deployment "channels" for engineering cloud applications. This is in contrast with the "syndicated multi-channel cloud service delivery model" approach in Figure 1.6(b) that allows an CSBA to be constructed through several alternative "channels" of vertical and horizontal cloud service compositions.

Figure 1.6: Engineering CSBAs using (a) monolithic cloud services vs. (b) Syndicated Multi-channel Cloud Service Delivery Model



(a) Monolithic Cloud Service Delivery Model

(b) Syndicated Multi-channel Cloud Service Delivery Model

The syndicated multi-channel cloud service delivery model supports the engineering of an CSBA using different alternative SaaSs. At the same time, it allows multiple (and possibly composed) PaaS/IaaS deployment options for a given SaaS. Traceability is also well-supported between CBSA-layer design decisions and the configuration of the underpinning SaaSs, PaaSs, and IaaSs.

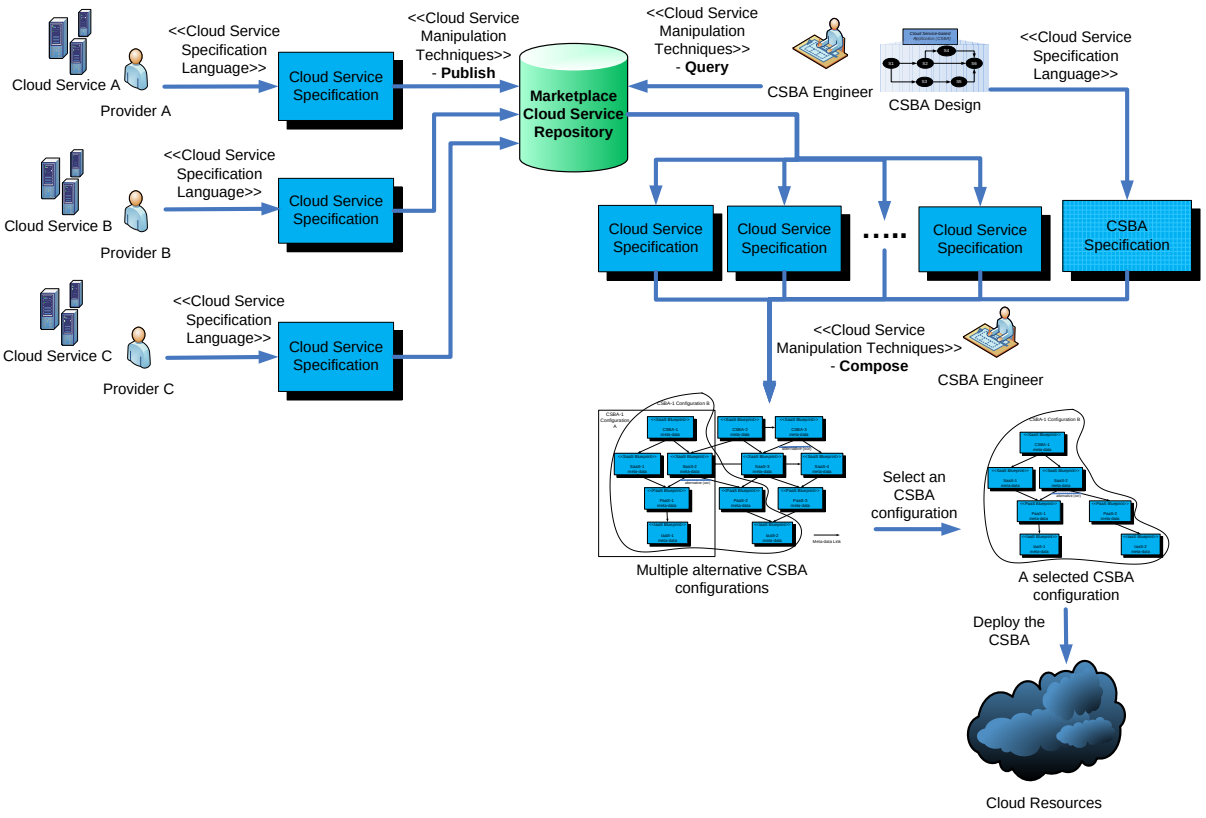
An CSBA Engineering Lifecycle

Based on the syndicated multi-channel cloud service delivery model, the authors in [Papazoglou & van den Heuvel, 2011] has proposed a lifecycle for engineering CSBAs. Figure 1.7 illustrates an excerpt of this lifecycle focusing on the design and configuration of an CSBA. There are three actors in the lifecycle, which will be introduced in the following:

- *Marketplace*: stores and manages the cloud service specifications in its cloud service repository. It also enables the publishing and querying for cloud service specifications. Some marketplaces support also the procedures of purchasing and contracting cloud services between the provider and the consumer, e.g. the SAP Service Marketplace [SAP A.G., 2011] or the 4caaSt marketplace [Gómez et al., 2011].

- *Cloud Service Providers*: specify their cloud services using a uniform *cloud service specification language*. Then, they can use a *cloud service manipulation technique* to publish the cloud service specifications to a marketplace.

Figure 1.7: CSBA Engineering Lifecycle as proposed in [Papazoglou & van den Heuvel, 2011]



- *CSBA Engineers*: Since an CSBA can be considered as a composite SaaS, an CSBA engineer can use the *cloud service specification language* to specify his CSBA as a cloud service specification. Then, by using the *cloud service manipulation techniques*, he can query the marketplace to retrieve the cloud service specifications required by his CSBA. The *cloud service manipulation techniques* also support the CSBA engineer with the composition of the cloud service specifications. Composing cloud service specifications results in several alternative CSBA configurations. Definition 1.3 defines an CSBA configuration as a composition of all the cloud service specifications required for configuring the deployment environment of the CSBA. Then, an optimum CSBA configuration is selected based on predefined criteria, e.g. licensing or cost. Finally, the selected CSBA configuration serves as a deployment manifest for configuring the deployment environment for the CSBA.

Definition 1.3 (CSBA Configuration) *An CSBA configuration is a composition of all the cloud service specifications required for configuring the deployment environment of an CSBA.*

The design and configuration of CSBAs in this lifecycle emphasizes the need of a uniform *cloud service specification language* across all three layers of the cloud stack, so that cloud service specifications created by various providers can be seamlessly assembled into a complete CSBA configuration. Since an CSBA can be considered as a composite SaaS, the *cloud service specification language* should also support the specification of an CSBA. Furthermore, as cloud services and CSBA could be specified in a uniform way, the lifecyle also signifies the need for a set of *cloud service manipulation techniques* that support the publishing, querying, and composing cloud service specifications with aim to configure an CSBA.

The research in this thesis is based on the CSBA engineering lifecycle proposed in [Papazoglou & van den Heuvel, 2011] with the focus on the design and configuration phase of an CSBA. It has been motivated by the need of a uniform *cloud service specification language* and a set of *cloud service manipulation techniques*.

1.3 Problem Definition

Most of the existing languages for specifying cloud services concentrate only on the IaaS layer, the Open Virtual Format (OVF) is a standard for specifying the packaging and distribution information of an IaaS [DMTF, b], and the Open Cloud Computing Interface (OCCI) is a standard for specifying the management API of an IaaS [OCCI-Working Group, 2011]. On the PaaS layer, there are only a few proprietary specification languages based on an ad-hoc XML definition supported by the commercial cloud vendors , e.g. the AmazonBeanstalk provided by Amazon for the Amazon WS, an ad-hoc XML format provided by Microsoft for the Microsft Azure, etc. On the SaaS layers, apart from the existing well-known languages for specifying Web Services, i.e. the WS-* stack, some initiatives have been proposed to support the automatic deployment and provisioning of SaaS applications, e.g. the Cafe template [Mietzner et al., 2009] or the OASIS's Topology and Orchestration Specification for Cloud Applications (TOSCA) [OASIS, 2013]. *As a summary, we observe that existing languages for specifying cloud services are restricted for one particular layer of the cloud stack.*

Another shortcoming of the existing cloud service specification languages is that they aim to support only the specification of some particular aspects of a cloud service. For instance on the PaaS/IaaS layer, they mostly capture only the technical details of a PaaS/IaaS to support its automatic deployment and provisioning. On the SaaS layer, except some new initiatives like TOSCA [OASIS, 2013] that also aim to support the automatic deployment and provisioning of a SaaS, the other specification languages mostly target the operational and performance description for the SaaS consumers. Furthermore, *existing cloud service specification languages do not aim to assist the CSBA engineer with techniques to publish, query, and compose cloud service specifications across several vendors*. Although there already exists a number of data manipulation

techniques, e.g. XQuery [W3C, 2013] and XPath [W3C, 1999a] for querying and manipulating XML data, SQL for querying and manipulating data stored in a database, SPARQL [W3C, 2008] and SPARQL-Update [W3C, 2012] for querying and manipulating data in an OWL data model, etc., there has not been any data manipulation technique that targets the CSBA configuration using cloud service specifications, to the best of our knowledge.

***Problem Definition**:*

- *There is a lack of a uniform specification language for cloud services across all three layers of the cloud stack.*
- *There is a lack of techniques for publishing, querying, and composing cloud service specifications with aim to support the configuration of an CSBA.*

In the next section, the problem definition will be tackled by a concrete research goal, which will then be decomposed into number of research questions.

1.4 Research Goal and Questions

To tackle the problem definition presented in the previous Section 1.3, we present in the following the research goal of this thesis

*The **Research Goal** of this thesis is twofold:*

- *G1: To develop a uniform specification language for cloud services across three layers of the cloud stack.*
- *G2: To develop techniques associated with the cloud service specification language to support the publishing, querying, and composition of cloud service specifications.*

Our research goal will be further scrutinized through a number of discrete research questions that initially led the way how the research in this thesis has been conducted:

Research Questions:

- **RQ1**: What is the *state-of-the-art* in specification language support for cloud services and what are their strengths and shortcomings?
- **RQ2**: How can we *design* and *validate* a uniform specification language for cloud services across all three layers of the cloud delivery stack?
- **RQ3**: How can we *design* and *validate* a set of manipulation techniques for publishing, querying, and composing cloud service specifications?

In the next section, we present the research methodology that has been adopted from existing literature to guide us to systematically answer the research questions and ultimately achieve our research goal.

1.5 Research Methodology

In this section, the research methodology used to systematically answer the research questions will be reported. As mentioned from the beginning of the chapter, we position our research within the domain of engineering CSBAs, which basically falls under the broader scope of Information System research[6]. Providing CSBA engineers with a cloud service specification language and a cloud service manipulation technique conceptually belongs to the design science [Hevner et al., 2004]. As the result, we have followed the design science methodology for information system research in [Hevner et al., 2004] and structured our research in the following four main steps

Step 1: Problem Definition

Before conducting any research, one has to understand clearly the problem that needs to be targeted. By identifying the research context and the motivation, we have articulated a concrete problem definition in Section 1.3. The research goal has been derived to target the problem definition and then it is decomposed into a set of research questions in Section 1.4. The research questions initially determined the direction for us to conduct this thesis.

Step 2: Literature Review

[6]In a general term, an Information System (IS) is any combination of information technology and people's activities that support operations, management and decision making (Wikipedia: `http://en.wikipedia.org/wiki/Information_system`). O'Brien and Marakas have defined in [O'Brien & Marakas, 2009] the five basic components of an IS: Personnel, Hardware, Software, Networks, and Data. Engineering CSBAs comprises of activities regarding the Software, hardware, network and data, and thus belongs to this research scope

Our objective in this step is to study extensively the existing approaches for specifying cloud service and manipulating cloud service specification data. It is crucial to understand the scope, purpose, and limitation of existing languages to discover the reuse opportunities of existing features. Chapter 2 will report the result of this step with a comprehensive comparison of existing approaches followed by an evaluation of their strengths and shortcomings.

Step 3: Solution Design

The shortcomings of existing approaches for cloud service specification and manipulation have given us the direction to design the solution. Chapter 3 presents an overview of our solution that comprised of a cloud service specification language and a cloud service manipulation technique. The solution will be explored and illustrated through a running example, which is borrowed from a real-world scenario that has been co-developed with our industry partners in the 4caaSt project [European Comission, 2010]. Then in Chapter 4, the cloud service specification language is introduced in detail with an abstract language model followed by its formalization in a formal knowledge model. Lastly, Chapter 5 introduces the cloud service manipulation technique as a set of operators for manipulating and cross-relating cloud service specification data.

Step 4: Validation and Evaluation

One of the most significant task in design science [Hevner et al., 2004] is the validation of the result to ensure its applicability in the real world. Regarding this task, we have performed several different activities for validating our solution:

1. The **technical soundness** of the cloud service specification language is guaranteed by its mapping to a formal knowledge representation model described in RDF/OWL [McGuinness & van Harmelen (Eds.), 2004]- a well-established standard for describing Internet resources and semantic web. Based on this formal knowledge model, the operators defined by the cloud service manipulation technique have been formalized using SPARQL [W3C, 2008] and SPARQL-Update [W3C, 2012] operations, which are standards for manipulating RDF/OWL knowledge model. Using well-defined and widely-accepted standards for knowledge representation and manipulation ensures the logical consistency in our proposed solution.

2. The **usability** of our approach is demonstrated by using a running example throughout the chapters of the thesis. This example is borrowed from a real-case scenario that has been developed by the 4CaaSt community as one of the three validation scenarios in the 4CaaSt project [European Comission, 2010]. Reusing

a scenario from the 4caaSt community shows that our solution solves a real-case defined by a group of cloud computing practitioners.

3. The **technical feasibility** of our approach is proved by an internal "proof-of-concept" prototype. This prototype will be reported in Section 6.1.

4. Within the 4caast community [European Comission, 2010], the proposed cloud service specification language and manipulation technique have been adopted as one of the core contributions of the 4caaSt project [Gómez et al., 2012]. Various industry prototypes have been developed based on the concept of our solutions. Future cloud products are also envisioned in this direction. In Section 6.2, we will report the activities in the 4caast project to exhibit the **practical validity** of the Blueprint Approach.

Also in this step, the final task is to evaluate how our solution can overcome the shortcomings identified in Step 2, and thus can be approved as an advanced contribution to the domain of engineering CSBAs. In Section 6.3, we will evaluate our approach by comparing it with existing approaches.

1.6 Contributions

To answer the research questions in Section 1.4, our work revolves around the concept of *Blueprint* whose definition is given in Definition 1.4.

The **contribution** of this thesis is to provide a *Blueprint Approach* for engineering CSBAs that includes the following components:

- **C1**: A well-defined *Blueprint Specification Language* (BSL) that provides a means for cloud service providers to abstractly (i.e., independent of implementation) and unambiguously specify a cloud service in a blueprint.

- **C2**: A set of *Blueprint Manipulation Techniques* (BMTs) for publishing, querying, and composing blueprints with aim to support the design and configuration of an CSBA.

Contribution C1 is the answer of the research question RQ2 and contribution C2 is the answer to the research question RQ3. During the development of both C1 & C2, we take into account the state-of-the-art analysis of existing specification languages for cloud services. Hence, with the outcome of the two contributions C1 & C2, we have also answered the research questions RQ1.

The *Blueprint Approach* has been developed during our participation in the European Commission's (EC's) 4CaaSt project [European Comission, 2010], which has the goal of creating an advanced cloud platform that supports the optimized and flexible

hosting of Internet-scale multi-tier applications. The 4CaaSt platform will contain the features necessary to facilitate the programming of rich applications and enable the creation of a true business ecosystem where applications, platforms and infrastructure from different providers can be traded, customized and combined. Real-world scenarios developed within the 4CaasT project will be used for purpose of validating the applicability of the *Blueprint Approach*.

Definition 1.4 (Blueprint) *A Blueprint is defined as a uniform, implementation-agnostic specification of a cloud service on any layer of the cloud stack, i.e. SaaS, PaaS or IaaS. There are three types of blueprints: SaaS blueprints are the SaaS specifications, PaaS blueprints are the PaaS specifications, and IaaS blueprints are the IaaS specification. A blueprint contains the following inter-related information sets [Papazoglou & van den Heuvel, 2011]:*

- *Operational service description: This information set focuses on the description of functional characteristics of a cloud service such as service types, messages, interfaces and operations, namely, the service's signature.*
- *Performance-oriented service capabilities: This information set includes key performance indicators (KPIs) associated with a cloud service. Typical examples of quantifiable KPIs are upper and lower performance response time ranges and service availability, throughput, delivery, latency, bandwidth, MTBF (Mean Time Between Failure), MTRS (Mean Time to Restore Service), and so on.*
- *Resource utilization: This information set describes the physical infrastructure and resources that are required to run a cloud service. In general, it expresses the workload profile including average and peak workload requirements. For instance, a cloud service provider may declare specific technical features that must be taken into account for his cloud service to operate properly, e.g., the server (disk I/O and network) bandwidth required for true on-demand delivery of streaming media, such as video and audio files. This set can express information packaged using existing standard like the DMFT's Open Virtualization Format (OVF) [DMTF, b].*
- *Policies: This information set prescribes, limits, or specifies any aspect of a business agreement that is required to use a particular cloud service. It is typically annotated with service level agreements (SLAs) and compliance rules and includes amongst other things security, privacy and compliance requirements.*

1.7 Assumptions and Limitations

Engineering CSBAs is a relatively new research domain and still contains many challenges that our blueprint approach cannot support. It is inevitable that our contributions still contain a number of assumptions and limitations, which will be explained in the following:

- *A1- Assumption regarding cloud application development*: In this thesis, we assume that cloud applications are developed following the CSBA style that reuses and composes third-party cloud services across all three layers of the cloud stack, i.e. SaaS, PaaS, and IaaS.
- *A2- Assumption regarding application migration*: When talking about migration, we assume that an application needs to be (fully or partly) migrated to the cloud. It means that we need to re-engineer the application following the CSBA style.
- *L1- Limitation of the BSL*: The BSL allows for specifying only certain information sets of a cloud service. These information sets have been selected as the most significant aspects of a cloud service specification from existing literature. These information sets include, for instance, the operational description, performance indicators, policy description, resource consumption, and resource requirements of a cloud service.
- *L2- Limitation of the BMTs*: The BMTs introduce only the techniques needed at the design time of an CSBA. In fact, we will only show how these techniques are used within the design and configuration phase of the CSBA engineering lifecycle introduced in Figure 1.7. Further supports at the runtime of an CSBA is out of scope of the BMTs.
- *L3- Dependency of the BMTs on the BSL*: The BMTs have been developed based on the BSL. Hence, an implementation of the BMTs works only on a concrete representation of the BSL.

1.8 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 presents our literature survey in specification languages and manipulation techniques for cloud services. Existing approaches will be analyzed and compared to identify their strengths and shortcomings as well as the reuse possibilities for our solution.

Chapter 3 aims to provide the readers the overview of the *Blueprint Approach* including its two components: the introduction of the BSL and the associated BMTs. This

chapter also contains the basic structural definition of a blueprint, its elements, and its dependency links. We will also explain in this chapter how the two components of the *Blueprint Approach* can assist the CSBA engineer within the design and configuration phase of the CSBA engineering lifecycle introduced in Figure 1.7. A running example is introduced in this chapter that will be used throughout the thesis to exemplify the use of the *Blueprint Approach*.

Chapter 4 targets the first component of the *Blueprint Approach* and introduces the *Blueprint Specification Language* (BSL). This language provides a common syntax for specifying cloud services across all three layers of the cloud stack, i.e. SaaS, PaaS and IaaS.

Chapter5 targets the second component of the *Blueprint Approach* and introduces a set of *Blueprint Manipulation Techniques* (BMTs) to support the publishing, querying, and composition of cloud service specifications across several providers. The BMTs have been developed as a set of operators that work on blueprints created by using the BSL.

Chapter 6 reports our efforts in validating the feasibility and applicability of the *Blueprint Approach*. Our validation activities include (1) a self-developed "proof-by-construction" implementation of the BSL and BMTs, and (2) our participation in developing an industry prototype within the 4caaSt project based on the concepts of the BSL and BMTs. Afterwards, the evaluation of the BSL and BMTs is presented.

Chapter 7 summarizes the thesis with the emphasis on its advanced contributions for the domain of engineering CSBAs. The limitations of this thesis are reviewed here to derive future research directions that may be tackled by interested readers.

CHAPTER 2

STATE-OF-THE-ART ANALYSIS

This chapter reviews the state-of-the-art in specification languages and manipulation techniques for cloud services. In Section 2.1, existing cloud service specification languages are reviewed and evaluated. These languages have been collected as related work throughout our long-term research and collaboration with both academic and industry partners in the cloud computing domain. We believe that they already represent the most state-of-the-art language supports for cloud service specification. Regarding the techniques to support cloud service manipulation we observe that there has not been any work in this direction. Hence in Section 2.2, we review the generic approaches in data manipulation, mapping and transformation, which gave us the direction for developing the manipulation techniques for cloud service specifications.

2.1 Cloud Service Specification Languages

[1] The concept of Blueprint is defined as a specification of a cloud service that abstracts away from all specific technical details and complexities to facilitate the CSBA developers with the selection, customization and composition of cloud services across various vendors. In this sense, the concept of blueprint is somewhat similar to the classical "Abstract Data Type" concept in the 70's that facilitates the abstraction of complex data objects for the development of a reliable, efficient and flexible software. Guttag was the first one who recognizes the significance of a precise specification of Abstract Data Types. He proposed an algebraic specification language for abstract data types in his earliest work in [Guttag, 1977].

[1] The result presented in this section has been partly published in [Nguyen et al., 2012c] [Taher et al., 2012]

The need for a formal specification language for software components has also been recognized in the component-based software development. Much of the work has focused on the functional component specification with aim to facilitate the component retrieval and reuse (a survey of work in this direction can be found in [Mili et al., 1995]). However, these languages usually take into account only functional specification. Additional languages like NoFun [Franch et al., 1999] have appeared somewhat later to facilitate the non-functional specification of components that enables a more precise component retrieval and matching.

Within the SOA domain, there exists already a large body of work in defining a standardized specification languages for software services, e.g. the most prominent and widely accepted standard is the W3C standard WSDL [W3C, 2011]. These standardized languages have a high potential reuse for specifying SaaSs, since cloud services are the logical extension of software services in a SOA systems[2]. They may contribute different aspects for cloud service specification on the SaaS layer.

In Section 2.1.1, we introduce the criteria used to evaluate the existing specification languages for cloud services on all three layers of the cloud stack. In section 2.1.2, 2.1.3, and 2.1.4, we review existing specification languages that target the cloud services on each specific layer of the cloud stack, i.e. IaaS, PaaS, and SaaS respectively. Existing specification languages for software services in the SOA domain are also reviewed in Section 2.1.4 since they may have high potential reuse for specifying SaaSs. Finally, Section 2.1.5 summarizes our analysis in this section.

2.1.1 Evaluation Criteria

In the subsequent sections, each existing specification language for cloud services will be reviewed and evaluated using a set of predefined criteria. The evaluation criteria presented in this section is an amalgamation of criteria used in existing surveys [Sun et al., 2012a] [Papazoglou & Vaquero, 2012] of cloud service specification languages. The motivation of selecting these criteria is twofold: (1) to be able to evaluate specification languages for all three cloud layers, and (2) to cover the multi-facets of a language, e.g. its purpose, target audience, maturity, etc. With this set of criteria for the evaluation, it is easier to estimate the reuse effort in case we would like to reuse an existing language for developing a uniform cloud service specification language.

- Coverage: The language allows for specifying the business or technical data or both of a cloud service.

[2]Whilst SOA puts focus on applying the service orientation principles on the application layer, e.g. by composing loosely-coupled software services into an SBA, the cloud environment extends the service orientation concept to the entire IT computing stack, i.e. across all three layers: application, platform and infrastructure layers. The cloud promotes the reuse and composition of cloud services on all three layers to build an CSBA. More details can be found in the discussion about the intersection of SOA and cloud computing in Section 1.1.3

  - Business Data: specify the Capability, Service Level Agreement (SLA), Policy, Business Rules and Compliance, Licensing, etc., of a cloud service
  - Technical: specify the Operational Description, Quality-of-Service (QoS), Elasticity, Resource Utilization, Deployment Environment, etc., of a cloud service.

- Intended Users: We distinguish only the two users of a cloud service specification language: Cloud Service provider and Cloud Service Consumer. This is because our work is grounded on the CSBA engineering lifecycle (introduced in Section 1.2) that involves only the Cloud Service Providers and CSBA Engineers. An CSBA engineer is considered as a cloud service consumer as he is only interested in reusing and composing cloud services within the lifecycle. A cloud service consumer could also be an end-user who is only interested in using a single cloud service.

- Purpose: Cloud service specifications described by a language aim to support particular phases of the cloud service lifecycle

  - Service Design: The specification serves as a design of a cloud service.
  - Service Matching & Discovery: The specification can serve as a cloud service request that can be matched against the specification of other cloud services.
  - Service Composition: The specification aims to support the composition of cloud services.
  - Service Binding: The specification contains also the binding details for consumers to interact with a cloud service.
  - Service Implementation: The specification contains also details that guide the implementation of a cloud service.
  - Service Deployment & Provisioning: The specification contains the requirements of the deployment environment and resource provisioning at runtime of a cloud service.

- Representation: A language may provide different representation techniques. Some languages provide concrete representation techniques like an ad-hoc XML template or a proprietary template format. The others introduce only abstract reference model. Other representation techniques include the use of a taxonomy of attributes, a hash table, or an existing formal specification language.

- Maturity: We classify a language whether it is an academic proposal, a proprietary language of a vendor that might have already been used in a product, or a standard.

- Extensibility: A language could provide an explicit extension point or it is unknown.
- Features: This criterion indicates the specific goals or features of a language, e.g. to support interoperability, to support a Model-driven Engineering approach, to target a specific domain, or to increase usability, etc.

2.1.2 IaaS Specification Languages

The ability of manipulate, integrate and and orchestrate the deployment of IaaS resources for cloud application development has been proposed in the early time of cloud computing, e.g. through the "Sky Computing" [Keahey et al., 2009] and "Intercloud" [Buyya et al., 2010] proposals. However these proposals falls short of proposing a solution for the problem at hand due to a lack of standardized language to specify the IaaS resources. Since then, much of the recent work in cloud computing specifically focus on supporting a common, standardized specification for IaaSs.

In this section, we review and evaluate the existing specification languages for IaaS. The section is divided into two parts based on the two most common representation techniques provided by these languages. We observe that one group of the IaaS specification languages is based on a template structure which is either a well-defined standard, e.g. the OVF template [DMTF, b], or a proprietary template of a vendor, e.g. Amazon Formation. The other group of IaaS specification languages follows the model-driven engineering approach by specifying IaaS using models and model transformations.

Template-based IaaS Specification Languages

The Distributed Management Task Force (DMTF) group[3] has published open standards such as the Open Virtualization Format (OVF) [DMTF, b], to provide a packaging and distribution format for virtual appliances (i.e. virtual machines hosting a complete software/middleware stack), and the Virtualization Management (VMAN) [DMTF, a] specifications that address the management lifecycle of a virtual environment to help promote interoperable cloud computing service. The OVF [DMTF, b] is considered nowadays as an open standard for packaging and distributing virtual appliances. It contains a set of XML templates (conforming to predefined XSDs) to support the specification of either the offering of an IaaS provider or the infrastructure resource requirements of a SaaS or PaaS provider.

The work in [Galán et al., 2009] is grounded on the OVF template and proposes a service definition language for IaaS provider to configure the deployment of their

[3] `http://dmtf.org/`

IaaSs in federated infrastructure clouds. The proposed service definition language extends the OVF to support the configuration of an IaaS with Key Performance Indicators, deployment time parameters (e.g., hostnames, IP addresses and other application specific details) for the virtual appliances[4], runtime elasticity specification, and public network specification. The authors in [Rodero-Merino et al., 2010] reuses this service definition language proposal as the common IaaS description in an service lifecycle management system that enables the management of IaaSs on top of several federated infrastructure clouds. Central to this system is the Service Abstraction Layer (called Claudia), which automates the deployment and runtime scaling of IaaSs described in the aforementioned service description language.

Similar to OVF-based approaches, the Solution Deployment Descriptor (SDD) template [OASIS, 2008] proposed by OASIS defines an XML schema to describe the characteristics of an installable unit (IU) of software that are relevant for core aspects of its installation, configuration, and maintenance. The benefits of this work include: the ability to describe software solution packages for both single and multi-platform heterogeneous environments, the ability to describe software solution packages independent of the software installation technology or supplier, and the ability to provide information necessary to permit full lifecycle maintenance of software solutions.

Chieu et al. introduce in [Chieu et al., 2010] a 3-tier cloud provisioning system to simplify the deployment of applications on an IaaS cloud. Within this system, they introduce the concept of "composite appliance" to automate the deployment of complex applications. A composite appliance is a collection of individual appliance images that are designed to work together with specific configurations. A composite appliance is specified using an XML descriptor file and thus considered as a template-based IaaS specification language provided by this cloud provisioning system.

InterCloud [Bernstein et al., 2009] is a federated Cloud Computing Environment that enables the utilization of multiple infrastructure clouds for scaling purposes. Document [Bernstein et al., 2009] targets the interoperability between the federated clouds by providing a collection of proposals for "InterCloud" protocols and formats. One of the InterCloud proposals is the use of Extensible Messaging and Presence Protocol (XMPP) as the communication protocol between the cloud providers and the Intercloud Directory and Exchange, which plays the role as the mediator within the InterCloud [Bernstein & Vij, 2010]. Furthermore, to facilitate the federation of cloud resources among the cloud provider, the authors in [Bernstein & Vij, 2010] also propose the use of the Resource Description Framework (RDF) [Manola & Miller, 2004] and an ontology of cloud computing resources as a shared resource catalog across heterogenous cloud providers.

[4] i.e. software components (e.g., web/application servers, database, operating system) running on the VMs

In practice, an attempt to provide a JSON template[5] for using cloud services is available from Amazon through their AWS CloudFormation offering [Amazon,]. This template provides AWS developers with the ability to specify a collection of AWS cloud resources and the provisioning of these resources in an orderly and predictable fashion. Nevertheless, this template works only for AWS cloud platform and infrastructure resources and thus lacks interoperability. Another template-based support that enables the orchestration of tasks for managing virtual machines is VMware vCenter Orchestrator [VMWare,]. Nevertheless, both the AWS CloudFormation and the VMware vCenter Orchestrator work only on their own IaaS resources, i.e., Amazon's cloud infrastructure resources and VMware vCenter Servers respectively, and thus lacks interoperability across IaaS providers.

Model-driven Engineering Approach

Model-driven engineering approaches have also been used to support the specification and automatic deployment of IaaSs on the cloud. In general, model-driven engineering deals with the provision of models, model transformations and code generation for software development [Kleppe et al., 2003]. Model-driven engineering approaches use model as the first-class entity during the whole lifecycle of specifying, configuring and deploying an IaaS. By adopting a model-driven engineering approach, an IaaS provider can follow the guidelines to capture various aspects of an IaaS, e.g. operational, QoS, policy, resource description, using models, which can then be translated via standardized transformation rules into concrete deployment artifacts for a particular platform.

To unlock the vendor lock-in problem concerning the APIs, the Open Grid Forum's Open Cloud Computing Interface (OCCI) working group (www.occi-wg.org) has been developing a uniform API specification for remote management of Cloud Computing infrastructure [OCCI-Working Group, 2011]. This API specification supports the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring. The scope of this specification defines all the high-level functionality required for the life-cycle management of virtual machines (or workloads) running on virtualization technologies (or containers) supporting service elasticity. At this stage, the API specification has been proposed in terms of abstract models without any concrete implementations or tool support.

Apart from the API specification, model-driven engineering approaches have mostly been used for automating the deployment of IaaSs. For instance, Konstantinou et al. propose in [Konstantinou et al., 2009] a virtual appliance model, which treats virtual images as building blocks for IaaS composite solutions. Virtual appliances are composed into virtual solution model and deployment time requirements are then determined in a cloud-independent manner using a parameterized deployment plan.

[5]JavaScript Object Notation, a light-weight data interchange format: `http://www.json.org/`

Collazo-Mojoca et al. present their idea of a virtual environment in [Collazo-mojica et al., 2010] as an abstract model built on top of a group of independent virtual appliances. The aim of this model is to provide an uncomplicated model for designing, configuring and deploying an IaaS solution. The model allows for specifying independent virtual appliances with a service endpoint as the connection between two appliances. Once the user is finished with the design and configuration of the virtual environment model, the model can be persisted in an XML file that can later be deployed on a cloud environment.

Other approaches

SmartFrog [Goldsack et al., 2009] is a declarative configuration framework that aims to simplify the design, deployment, and management of distributed systems. Components of a distributed system managed by SmartForg can be specified using a hash table. The specification details in the hash table will be processed by SmartForg to properly configure the components. SmartFrog has been released as an open source solution since 2003.

Cloud# [Liu & Zic, 2011] is a formal specification language for modeling an IaaS cloud wtih aim to provide transparency of the IaaS cloud to the consumers. Cloud# is provided with a formal grammar which can be used by IaaS consumers to reason about how services are delivered in an IaaS cloud. The goal of providing a high transparency to IaaS consumers is to convince them to move business-critical applications to the IaaS cloud.

Summary and Evaluation

Table 2.1 summarizes the survey in this section with a comparative evaluation of the existing IaaS specification languages. We observe that there has been already a large body of work in IaaS specification languages with aim to support the IaaS providers in designing, deploying, and provisioning their IaaSs. These languages allow for specifying only the technical details of the deployment environment of an IaaS, i.e. by adopting the OVF as the standard for specifying the virtual appliance packaging and distributions. We conclude that existing IaaS specification languages based on the well-defined OVF standard, e.g. the one defined in [Galán et al., 2009], have the high potential reuse for defining an IaaS blueprint. However, they should be combined with other languages to specify the business information, policy, and resource consumption of an IaaS.

2.1.3 PaaS Specification Languages

The Service-Oriented Cloud Computing Architecture (SOCCA) proposed in [Tsai et al., 2010] allows developers to build applications within an integrated

Approach	Coverage	Intended Users	Purpose	Representation	Maturity	Extensibility	Features
[Galán et al., 2009] & [Rodero-Merino et al., 2010]	Technical (deployment environment, elasticity rules)	IaaS provider	Service Deployment & Provisioning	XML Template	used in Claudia, an open source Implementation (Affero GPL licensing), within the EU-project RESERVIOR	n/A	IaaS language is an extension of OVF with KPI, Deployment Time Parameters, Elasticity Specification, and Public Network Specification
SDD [OASIS, 2008]	Technical (deployment environment)	IaaS provider	Service Deployment & Provisioning	XML Template	OASIS Standard	n/A	Software Distribution and Deployment
Composite Appliance in [Chieu et al., 2010]	Technical (deployment environment)	IaaS provider	Service Deployment & Provisioning	XML Template	used in IBM Research Compute Cloud RC2 environment	XML extension	n/A
Intercloud [Bernstein & Vij, 2010]	Technical	IaaS provider	Service Deployment & Provisioning	RDF model	Intercloud proposal	n/A	aims to federate resources of several IaaS clouds
Amazon CloudFormation [Amazon,]	Technical	IaaS consumer	Service Deployment & Provisioning	JSON Template	Proprietary Vendor Format	n/A	support also for configuring PaaS solutions of Amazon
Virtual Solution Model in [Konstantinou et al., 2009]	technical	IaaS provider	Service Design, Service Composition, Service Deployment & Provisioning	Virtual Solution Model built on a modeling platform, a component of the IBM RSA v7.5	IBM product	n/A	n/A
Virtual Environment in [Collazo-mojica et al., 2010]	technical	IaaS provider	Service Design, Service Composition, Service Deployment & Provisioning	Abstract Model for Service Specification, but then transformed into XML format	Academic Proposal with Prototype	n/A	n/A
OCCI [OCCI-Working Group, 2011]	Technical (interface specification)	IaaS provider	Service Management	Specification Model	Open Grid Forum (OGF) Recommendation	n/A	Standard API Specification for Cloud Service Management
Cloud# [Liu & Zic, 2011]	technical	IaaS consumer	Service Design, Service Deployment & Provisioning	Formal Specification Language	Academic Proposal	n/A	aims to provide more transparency of the IaaS to the consumers
SmartFrog [Goldsack et al., 2009]	technical	IaaS provider	Service Deployment & Provisioning	Ordered hash table	Academic Proposal	n/A	n/A

Table 2.1: Comparative Evaluation of IaaS Specification Languages

SOA framework. Cloud platform and infrastructure resources may be discovered by a Cloud Broker Layer and a Cloud Ontology Mapping Layer for deploying the application components. The multi-tenancy feature of cloud computing is also supported by SOCCA where multiple instances of platform resources can be provided to multiple tenants.

Windows Azure[6] is a Microsoft cloud platform that can be used to build, deploy,

[6]Microsoft Windows Azure: `http://www.windowsazure.com`

and host applications on Microsoft's data center. Services in Windows Azure are configured by two XML files [Microsoft MSDN, 2011]:

- The service definition file describes the service model. It defines the roles included with the service and their endpoints, and declares configuration settings for each role. The default extension for the service definition file is .csdef
- The service configuration file specifies the number of instances to deploy for each role and provides values for any configuration settings declared in the service definition file. The default extension for the service configuration file is .cscfg.

The CompatibleOne platform[7] is a cloud service broker that offers a simple and unique interface for discovering and deploying the needed cloud resources. Service requests are described in XML files named MANIFEST documents to capture the specific technical and business specifications. The MANIFEST document allows users to specify both the IaaS resources like the computing, network, or storage resources and the PaaS resources like a database or runtime container.

Cordys[8] is a cloud platform that specializes in Business Process Management (BPM) solution. The Cordys platform aims to support the design, execution, monitoring and improvement of business processes. The business processes in Cordys are modeled using the Business Process Model and Notation (BPMN). Services in Cordys are integrated using the WS-* stack to support the business process design and execution.

Amazon Elastic Beanstalk[9] is a PaaS solution that allows users of the Amazon Web Services (AWS) to quickly deploy their applications and seamlessly manage the underlying AWS resource during runtime. The AWSs that can be managed by Amazon Elastic Beanstalk include the Amazon Elastic Cloud Compute (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon Simple Notification Service (Amazon SNS), Elastic Load Balancing, and Auto Scaling. Amazon Elastic Beanstalk is built using familiar software stack to ensure the portability of users' applications. The API of Amazon Elastic Beanstalk is described using WSDL.[10]

Summary and Evaluation

Table 2.2 summarizes the survey in this section with a comparative evaluation of the existing PaaS specification languages. There are only a few specification languages that target the PaaS layer of the cloud stack. Most of them are proprietary languages provided by a commercial PaaS provider with aim to assist the PaaS consumers in composing and deploying the PaaS solutions. The representations of these languages

[7] `http://www.compatibleone.org/bin/view/Main/`

[8] `http://www.cordys.com/`

[9] `http://aws.amazon.com/elasticbeanstalk/`

[10] Amazon Elastic Beanstalk API Reference: `http://docs.aws.amazon.com/elasticbeanstalk/latest/api/Welcome.html`

Approach	Coverage	Intended Users	Purpose	Representation	Maturity	Extensibility	Features
Microsoft Azure	Technical	PaaS Consumer	Service Deployment & Provisioning	XML Format	Proprietary Format (Microsoft)	n/A	n/A
CompatibleOne	Technical	PaaS Consumer	Service Matching & Discovery, Service Deployment & Provisioning	XML files (named MANIFEST documents) to describe the needed services	Proprietary Format	n/A	n/A
Cordys	Technical	PaaS Consumer	Service Design, Service Composition, Service Deployment & Provisioning	Services are described by using the WS-* standards	Proprietary Format	n/A	a BPM platform for designing, executing, monitoring, and improving buisness processes
AmazonBeanstalk	Technical	PaaS Consumer	Service Deployment & Provisioning	using WSDL	Proprietary Format (Amazon)	n/A	a PaaS solution to deploy and manage applications on several AWSs, e.g. Amazon EC2, Amazon S3, etc.
Services in SOCCA [Tsai et al., 2010]	Technical	PaaS Consumer	Service Deployment & Provisioning	RDF/OWL	Academic Proposal	n/A	n/A

Table 2.2: Comparison between PaaS Specification Languages

are only proprietary template formats. This limits the interoperability between several PaaS providers and results in a vendor lock-in situation for the PaaS consumers. We see no potential reuse of any existing PaaS specification language. However, this survey has helped us identify the essential information that should be captured in a PaaS blueprint.

2.1.4 SaaS Specification Languages

The SaaS layer of the cloud stack is recognized as the intersection between the two research domains SOA and Cloud Computing[11]. Hence, existing service specification languages in SOA, e.g. the WS-* stack, may have a high reuse potential for specifying SaaSs.

SaaS specification languages in the SOA domain

The de facto standard in specifying services in the SOA domain is to use the Web Service Description Language (WSDL) [W3C, 2011]. It is an XML-based specification schema that allows users to specify the interaces, operations, message structures, data types of the message payload, binding protocols, and the endpoint addresses of a (Web) service. Using WSDL, a web service can be specified with multiple interfaces (portTypes), where each interface contains a set of operations and each operation is defined with a set of input and output messages. An interface can be mapped to several binding protocols and each binding protocol is defined for a particular endpoint address of the web service.

[11]Please refer to the discussion about the intersection of SOA and Cloud Computing in Section 1.1.3

WSDL has been recognized as the dominant standard for specifying web services, since it is backed by a strong industry support. However, WSDL focuses only on the operational specification of a single service. Hence, other specification standards have been proposed to complement WSDL such as the Business Process Execution Language (WS-BPEL) [OASIS, 2007] for composing web services and the WS-Policy [W3C, 2006] for specifying the policy description of a web service. As a result, these standards have collectively form a *Web Service technological stack* (or the WS-* stack).

WS-BPEL [OASIS, 2007] is an XML-based composition language for web services. A WS-BPEL specification sits on top of the WSDL specifications of other web services and defines a flow of interactions with these web services. WS-BPEL supports the basic constructs of a flow such as the sequence, parallel branches, conditional branches, etc. For each service interaction, WS-BPEL supports the definition of input and output messages by importing the WSDL of a constituting web service.

WS-Policy [W3C, 2006] supports the policy specification of a web service. Similar to other standards in the WS-* stack, it is also an XML-based language and built on top of the WSDL standard. Using WS-Policy, a policy profile can be defined for a web service, which contains a set of policy alternatives. Each policy alternative contains a set of policy assertions. Each policy assertion is specified for a WSDL operation to prescribe either a requirement or capability of its exchanged messages (e.g. the encryption, transport protocol, authentication, etc.) or a non-functional property related to its selection and usage (e.g. QoS property).

One of the shortcomings of the WSDL is that it supports only the structural specification of a web service. To complement WSDL, the semantic web community has introduced several initiatives to enrich a WSDL specification with more semantic information. For instance, the OWL-S [W3C,] is an OWL ontology that allows for describing the functional and operational semantics of a web service. The operational semantics is described in the sub-ontology "Service Profile" and the functional semantics is described in the other two sub-ontologies "Service Model" and "Service Grounding" in conjunction with a WSDL specification. Another initiative is the WSDL-S [W3C, 2005b] that supports semantic annotations directly on a WSDL specification. Lastly, the Web Service Modeling Ontology (WSMO) provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automatic discovery, composition and binding [W3C, 2005a].

It is worth to mention that apart from the languages that specifically target the Web Service technology there exist also other languages that aim for a more general specification of services in a SOA system, which will be mentioned in the following.

The W3C Unified Service Description Language (USDL) Incubator Group[12] has

[12] `http://www.w3.org/2005/Incubator/usdl/`

been formed to develop a general languages for specifying both the business and technical services. The outcome of this group is the Universal Service Description Language (USDL) [Internet of Services,], which aims to complement the technical service specifications, e.g. using WSDL, with business and operational information. The language can be used by both the service consumers and providers for different purposes: providers can design their services in a tradeable and consumable way whilst consumers can use the language for service discovery based on the required business capability. USDL is a generic language that aims for standardizing the service sector, and hence can be extended in several industry domains. It is delivered as a set of reference models that describe many business and technical aspects of a service.

The OASIS Reference Model for Service Oriented Architecture [OASIS, 2006] provides a reference model for building an entire SOA system. It aims to provide a common understanding of entities and their relationships in a SOA system and can be used to develop standards for a specific entities in SOA. Not like WSDL and WS-BPEL, this reference model is an abstract model that is not tied to any particular SOA technology and hence can serve as a common reference models across several SOA implementations. The OASIS reference model defines three conceptual views: service ecosystem, realization, and ownership, in which service specification falls under the realization view.

The CBDI-SAE Meta Model for SOA [Everware-CBDI,] is a meta-model defining the concepts in the CBDI Service Architecture & Engineering Reference Framework, an initiative of the Everware-CBDI consulting company[13]. Similar to the OASIS reference model, CBDI-SAE Meta Model for SOA aims to provide a consensus on the definition of a SOA system by conceptualizing the key components of many aspects of a SOA system. Different aspects of SOA are captured in different modeling packages: technology, organization, policy, business modeling, service modeling, deployment, runtime, etc. The CBDI-SAE Meta Model is technology-agnostic. It can be implemented using a concrete SOA technology, e.g. using the WS-* stack or REST.

It is interesting to see that both the OASIS Reference Model and the CBDI-SAE meta model recognize the significance of a reference model for SOA while remaining technology-agnostic. Both of them use the Unified Modeling Language (UML) [OMG, 2011] as the modeling and communication language for their models. Another initiative to define a reference model for SOA based on the UML is the Service-oriented Architecture Modeling Language (SoaML) specification [OMG, 2012] proposed by the Object Management Group (OMG). The SoaML specification provides a meta-model and an UML profile that cover both the business and technical specification of a service. Its main goal is to support the modeling and design of a service in a SOA system with a set of core attributes. SoaML can be extended with attributes related to a specific domain. For instance, the work in [Elvesæter et al., 2011]

[13] `http://everware-cbdi.com/home`

has proposed specific service attributes and modeling guidelines to use the SoaML for specifying services in the cloud domain. However, this set of attributes support only ofr specifying SaaSs and does not have the intention to go further down to the lower layers of the cloud stack.

Another approach for specifying SOA service in a technology-agnostic is to use the Abstract Service Description (ASD) meta-model introduced in [Andrikopoulos, 2010]. The ASD meta-model in this work is a specification schema for specifying an interface of a service including its structural, behavior and non-functional description. It aggregates the important concepts found in widely adopted technologies like WSDL and WS-BPEL, as well as from the high-level reference models like the OASIS reference model for SOA and the CBDI-SAE meta-model. However, this approach captures only the interface of a service and aims only to provide a mechanism to control the service evolution.

Finally, O'Sullivan proposes in his thesis [O'Sullivan, 2006] a taxonomy of non-functional properties for service specification. This work does not specifically target web services or services in a SOA system, but can be used for any types of services. Its aim is to provide a more precise matching and discovery of services based on their non-functional properties. The properties proposed in this work are domain-independent and can be used in different industry domains.

One of the biggest disadvantages of the SaaS specification languages in the SOA domain is that they do not aim to support the automatic deployment and provisioning of a SaaS. In the following, we present some initiatives in providing SaaS specification languages that aim to support the automatic deployment and provisioning of a SaaS.

Other SaaS specification languages

The Model Driven Engineering (MDE) research community has realized the benefit of combining MDE techniques with application development and suggested combining MDE with cloud computing [Brunelière et al., 2010]. As the article describes, there is no consensus on the models, languages, model transformations and software processes for the model-driven development of cloud-based applications. Following the MDE vision, the authors in [Hamdaqa et al., 2011] propose a meta-model that allows cloud users to design SaaS applications independent of any platform and build inexpensive elastic applications. From their point of view, a SaaS application should avoid the vendor lock-in problem concerning the underlying platforms. This meta-model support the description of the capabilities, technical interfaces, and configuration data for the virtualized infrastructure resources of the cloud application service. Similarly, the work in [Cai et al., 2009] presents a different customer-centric cloud service model. This model concentrates on aspects such as the customer subscription, capability, billing, etc., yet does not cover other technical aspects of the cloud services including the technical interfaces of the cloud services, the elasticity, the required de-

ployment environment, etc. Other existing models, e.g. in [Thrash, 2010], also lack a formal structure and dentitions (reducing their usability and reusability) or are not explicit and assume tacit knowledge.

Based on the IaaS specification language proposed by Galan et al. in [Galán et al., 2009], Chapman et al. defines in [Chapman et al., 2010] a manifest language called *Application Description Language* to serve as a contract between an application developer and an IaaS provider. In this contract, architectural constraints and invariants regarding the infrastructure resource provisioning for an application are specified and can be used for on-demand cloud infrastructure provisioning at run-time. The abstract syntax of the manifest language is modeled using the Essential Meta-Object Facility (EMOF), an OMG standard part of the Model Driven Architecture initiative[14]. Using the manifest language to specify the structure of an application, i.e., the application components and their required Virtual Execution Environments (VEE), the Reservoir architecture in [Rochwerger et al., 2009]can automatically provision the VEE instances that can run simultaneously without conflict on a federated cloud infrastructure of multiple providers. KPI monitoring mechanisms and elasticity rules in the manifest act as a contract that guarantees the required Service Level Agreement (SLA) between the application developer and the Reservoir architecture.

A cloud-agnostic middleware is introduced in [Maximilien et al., 2009] that can sit on top of many PaaS/IaaS offerings and enable a platform-agnostic SaaS development. They provide a meta-model for describing SaaS applications and their needed cloud resources, and APIs and middleware services for the deployment.

Sun et al. propose in [Sun et al., 2012b] the use of ontology to specify and match the resource requirements of a SaaS application with the available resources of the cloud infrastructure. They provide also a resource allocation support based on the ontology mapping technique.

The Cafe application and component templates [Mietzner, 2010] are a relevant approach for cloud-based SaaS development. Cafe provides an ad-hoc composition technique for application components and cloud resources following the Service Component Architecture (SCA). However, this approach requires SaaS developers to possess intricate technical knowledge of the application architecture and the physical cloud deployment environment to select and compose the right application components and cloud resources.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [OASIS, 2013] is an OASIS standardization initiative for describing the deployment topology of a composite application in the cloud environment. TOSCA defines generic templates for specifying nodes and node relationships in an application topology, where each node is defined explicitly with management operations.

[14]OMG MOF: `http://www.omg.org/mof/`

Furthermore, TOSCA introduces the concept of management plan, which orchestrates the management operations of the nodes using standards like WS-BPEL. Using TOSCA as the underlying specification of the deployment topology, the concept of a portable management of cloud service has been introduced in [Binz et al., 2012].

Summary and Evaluation

Table 2.3 summarizes our review in this section with a comparative evaluation of the specification languages for SaaSs in both the SOA and Cloud domains. When looking at the languages supported in the SOA domain, we find a large body of widely adopted standards for building SOA systems, e.g. the WS-* stack contains a set of W3C standards, the SoaML [OMG, 2012] is an OMG standard, and the CBDI-SAE reference model [Everware-CBDI,]. Most of them cover both the business and technical specification of a SaaS and aim to support all the phases of a service lifecycle. We see a great benefit to reuse these languages for specifying SaaSs. In the following we discuss our interests in reusing the existing SaaS specification languages:

- WS-Policy [W3C, 2006] has the high potential reuse for defining a policy description in a blueprint due to its wide adoption and simplicity.
- The ASD meta-model [Andrikopoulos, 2010] has the high potential reuse for defining the service interface of a SaaS blueprint. The reason of adopting this meta-model is that it has unified all the important concepts of the WSDL [W3C, 2011], WS-BPEL [OASIS, 2007], OASIS Reference Model for SOA [OASIS, 2006], and CBDI-SAE Meta-model for SOA [Everware-CBDI,], into a single meta-model for describing software service interfaces in SOA.
- SoaML [OMG, 2012] and USDL [Internet of Services,] have the high potential reuse for specifying the business information of a blueprint. However, we would rather consider them as the potential add-ons for our cloud service specification language.

Apart from the well-known standards for specifying Web Services, we observe a lot of interesting language initiatives in specifying the deployment environment of a SaaS application, e.g. Chapman et al. [Chapman et al., 2010], Cafe [Mietzner et al., 2009] and TOSCA [OASIS, 2013]. We draw our particular attention to TOSCA because of its support for specifying the deployment topology and management plan for a cloud application. These two features have not been supported by any other languages. However, TOSCA does not support the specification of cloud services on all three layers of the cloud stack and does not consider the composition of these specifications as the deployment topology of a cloud application. This is where our work is differentiated from TOSCA. Nevertheless, we see a great benefit of using TOSCA to generate a deployment configuration and management plan for an CSBA configuration with blueprints.

Approach	Coverage	Intended Users	Purpose	Representation	Maturity	Extensibility	Features
O'Sullivan [O'Sullivan, 2006]	Business	Service Consumer	Service Matching & Discovery	Attribute Taxonomy	academic proposal	n/A	Domain independent
USDL [Internet of Services,]	Business & technical	Service consumer, Service provider	Service Design, Service Matching & Discovery, Service Binding	Reference Metamodel	Standardization through a W3C Incubator Group	yes	Domain independent, Standardization in the service sector
WS-* stack including WSDL [W3C, 2011], WS-BPEL [OASIS, 2007], WS-Policy [W3C, 2006], etc.	Technical	Service consumer, Service Provider	Service Design, Service Matching & Discovery, Service Composition, Service Binding, Service Implementation	XML Template	W3C Standard	Basic XML extension	Intended for SOA & Web Service Development
CBDI-SAE Meta-model for SOA [Everware-CBDI,]	Business & Technical	Service Consumer & Provider	Service Design, Service Composition, Service Implementation, Service Deployment & Provisioning	Reference metamodel in UML	Company Proposal	n/A	Technology-independent, reference model for a SOA system
OASIS Reference Model for SOA [OASIS, 2006]	Business & Technical	Service Consumer & Provider	Service Design, Service Composition, Service Implementation	Reference metamodel in UML	Official OASIS Standard	n/A	Technology-independent, reference model for a SOA system
SoaML [OMG, 2012]	Business & Technical	Service Provider	Service Design	Reference metamodel in UML and a UML profile	OMG Specification	yes	Support for SOA and cloud services
Abstract Service Description metamodel [Andrikopoulos, 2010]	Technical	Service consumer, Service provider	Service Design	Reference metamodel in UML	academic proposal	n/A	support for service evolution control
Manifest Language in [Chapman et al., 2010]	Technical	SaaS Provider	Service Deployment & Provisioning	EMOF-metamodel	Academic Proposal	n/A	based on the IaaS specification language in [Galán et al., 2009]
Application Meta-model in [Hamdaqa et al., 2011]	Business & Technical	SaaS provider & Consumer	Service Modelling & Design, Service Deployment & Provisioning	Reference Meta-Model	Academic Proposal	n/A	n/A
Cafe template [Mietzner, 2010]	Technical	SaaS Provider	Service Deployment & Provisioning	Ad-hoc XML Template	Academic Proposal	n/A	n/A
TOSCA [OASIS, 2013]	Technical	SaaS provider	Service Deployment & Provisioning, Service Management	Ad-hoc XML Template	OASIS standard	n/A	with aim for portable management of application
Customer-centric model in [Cai et al., 2009]	Business	SaaS provider & consumer	Service Design, Service Matching & Discovery	Reference Model	Academic Proposal	n/A	n/A
Cloud-agnostic middleware [Maximilien et al., 2009]	Technical	SaaS provider	Service Deployment & Provisioning	Reference Metamodel	Academic Proposal	n/A	n/A
Ontology-based Resource Allocation [Sun et al., 2012b]	Technical	SaaS provider	Service Deployment & Provisioning	RDF/OWL ontology	Academic Proposal	n/A	n/A

Table 2.3: Comparative Evaluation of SaaS Specification Languages

2.1.5 Summary and Evaluation of Existing Cloud Service Specification Languages

In general, we observe that most of the existing specification languages are restricted only for cloud services on a particular layer of the cloud stack. Besides, they usually cover either the technical specification of a cloud service, e.g. packaging and distribution of virtual appliances, APIs, scalability, etc, or the business specification, e.g. policy, SLAs, etc. They fail to cover the holistic picture of cloud services across all three layers of the cloud stack.

Regarding the intended purpose of the language, the most common purpose is to support the specification of technical meta-data that serve for the automatic deployment and provisioning of a cloud service. Other purposes address the vendor lock-in problem by proposing a standardized language for specifying either the management API or the virtual appliance packaging information. However concerning the vendor lock-in there are still many unsolved compatibility issues beside the API compatibility, such as the data format, billing, metering, error handling, logging, or cloud management and administration, as pointed out by Vambenepe in [Vambenepe, 2009]. Service composition has not been recognized much as the purpose in an existing language. Especially, the idea of supporting the vertical composition of cloud services between two adjacent layers has not been found in any languages.

Extensibility is also very limited in existing specification languages, which prevents the combination of several languages.

As a conclusion, the state of the art analysis has identified the shortcoming that there is currently a lack of a uniform specification language that:

1. Can be used for cloud services on all three cloud layers.
2. Unifies all views on a cloud service, e.g. from the customer's view on the operational description, APIs and policy, to the developer's view on service composition, deployment environment and runtime provisioning issues.
3. Can assist the CSBA engineers to manipulate and compose cloud services specifications of several vendors to address their application needs.
4. Can easily be extended or incorporated with external languages or specification schemas
5. Aims to become an open standard for the cloud computing community

The derived issues above have served as the *requirements* for developing the concept of *Blueprint*, which will be introduced in the next Chapter 3. Subsequently, chapter 4 will introduce a specification language for the blueprint. The goal of the blueprints is to support the CSBA engineers in manipulating and composing appropriate cloud services for their CSBAs.

2.2 Cloud Service Manipulation Techniques

Recent specification languages that are based on standardization efforts like the OVF [DMTF, b], the OCCI [OCCI-Working Group, 2011], and TOSCA [OASIS, 2013], have proposed to use a common shared set of information to specify cloud services on a certain layer of the cloud stack. The adoption of these approaches would open the market and allow differentiation of cloud service offerings based on their different capabilities and service level agreements. Some languages have been provided together with a technique for manipulating the cloud service specifications, e.g. most of the model-driven engineering approaches provide the ability to edit and compose model entities. However, these techniques work only for cloud services on the same layer of the cloud stack. We observe the lack of a technique or toolset that supports the manipulation and cross-linking of cloud service specifications across all three layers of the cloud stack.

Cloud service manipulation techniques respond to the need to create, retrieve, update, and delete cloud service specifications that exist in a cloud service repository, which corresponds to the basic CRUD data manipulation operations. Furthermore, since cloud service specifications can be assembled to form a complete CSBA configuration, the manipulation techniques should also support the composition of cloud service specifications. Having the desired features of the manipulation techniques in mind, we review in the following the most prominent data manipulation approaches that work on multiple data models.

Most of the specification languages introduced in the previous Section 2.1 use templates or models to store the specification data. XML is the most commonly chosen template-based representation whilst the other model-driven approaches also support the transformation of specification models into XML documents. Hence, studying the existing support on manipulating XML documents is crucial as it promises the unified technique that works with several distinctive specification languages. In the following we review the existing support for manipulating an XML document:

- XSL Transformation (XSLT) [W3C, 1999b]: is a language for transforming an XML document to another XML document. The old XML document remains unchanged and a new XML document is created with the same data but a different schema. The transformation is driven by a set of template rules specified in an XSLT document.

- XQuery [W3C, 2013]: is a query language that provides the mechanisms to extract and manipulate data in XML documents. XQuery uses XPath [W3C, 1999a] to identify a certain position in an XML document, and defines the five operations FOR, LET, WHERE, ORDER BY, RETURN (FLWOR) to perform data extraction or manipulation. The semantics of the FLWOR operations is similar to a

SELECT-FROM-WHERE clause to perform a JOIN operation in SQL.

- XMLBeans[15] is a technology for manipulating XML documents by transforming it into Java types. By compiling an XML schema, XMLBeans enables access to instances of that schema following the JavaBeans-style with "get" and "set" methods.

In model-driven engineering, The QVT (queries/views/transformations) language [OMG, 2011] is a set of OMG standards that support the model transformation. The QVT language works with any models that conform to the Meta-Object Facility (MOF) 2.0 meta model[16]. As explained in its name, this language supports the querying of data in a model, the view on a model to create a new one, and the transformation of a model to another one.

When working with semantic data models described in RDF [Manola & Miller, 2004] and OWL [W3C, 2004], SPARQL [W3C, 2008] and SPARQL-Update [W3C, 2012] are the most prominent query and manipulation languages that operate on a RDF/OWL data model. SPARQL support SQL-like query operations such as SELECT (to retrieve data in a table format), CONSTRUCT (to retrieve data in the RDF format), ASK (to perform a simple true/false checking), and DESCRIBE (to extract a RDF sub-model from the RDF data model). SPARQL-Update supports manipulation operations such as the INSERT (to insert data or triples into a RDF model) and DELETE (to delete data or triples from a RDF model).

The work in [Pahl et al., 2009] introduces an ontology-based framework for modeling software architectures. The interesting part in this framework is that it does not only support the specification of an architecture style, but also provide a set of ontology-based operators for manipulating and composing architecture styles for defining the architecture of a software system. Since CSBA can also be considered as an architecture style that combines all the architectural principles and patterns from both SOA and Cloud domains, the operators for manipulating and composing styles in this work are very much related to our goal of defining a set of operators for cloud service manipulation.

The vision of generic model management has started in 2000 by Bernstein et al. [Bernstein et al., 2000]. By recognizing the problem of data heterogeneity and mismatch between different database schemas, Bernstein et al started a new research area in generic model management [Bernstein et al., 2000]. Models and model mappings are the two first-class objects in this research area. In this early work, the authors have sketched out the definition of models and model mappings, as well as the preliminary work on defining the operators for matching, differencing, and merging models. The Match operator has been recognized as the most important opera-

[15]XMLBeans: `http://xmlbeans.apache.org/`
[16]OMG MOF: `http://www.omg.org/mof/`

tor to create a model mapping. The first proposal for the Match operator appeared in [Madhavan et al., 2001] that specifically targeted the database schema matching problem. Since then, several improvements for the Match operator have appeared, such as the implementation of the Match operator in Rondo [Melnik et al., 2003] using a graph matching algorithm called "Similarity Flooding" or an approach to combine several schema matching approaches to implement the Match operator in the COMA system [Do & Rahm, 2002]. In his Phd thesis [Melnik, 2004], Melnik has developed a full set of generic model management operators based on the vision of Bernstein et al. This set of operators include both the elementary operators that work on a single model such as INSERT, DELETE, EXTRACT, UPDATE, and the complex operators that work between two models such as MATCH, MERGE, SPLIT, etc.

Our work towards the support of manipulation techniques for blueprints has been inspired by the generic model management operators defined by Melnik in [Melnik, 2004]. We have taken into consideration the applicability of these generic operators for cloud service specifications, selected a subset of them, and extended their definitions towards the requirements of the blueprint manipulation techniques, namely to support the publishing, querying, and composition of blueprints. Since the blueprints will be described using OWL [W3C, 2004], the formalization and implementation of these operators will rely on the use of SPARQL [W3C, 2008] and SPARQL-Update [W3C, 2012] - the existing well-established standards in querying and manipulating OWL models.

CHAPTER 3

THE BLUEPRINT APPROACH

This chapter introduces the *Blueprint Approach* as a support for the CSBA engineering lifecycle. By proposing the concept of *Blueprint* as an abstract, uniform specification of cloud services across all three layers of the cloud stack, the *Blueprinting Approach* is a novel solution that allows CSBA engineers to flexibly (re-)configure their CSBAs by assembling multiple alternative SaaS, PaaS, and IaaS specifications.

The chapter is organized as follows:

- In Section 3.1 we introduce the *Blueprint Approach* that includes the following two components: the Blueprint Specification Language (BSL) and the Blueprint Manipulation Techniques (BMTs). A motivating scenario that has been adopted from an industrial case study is also introduced in this section to motivate the use of the *Blueprint Approach* for engineering CSBAs.
- In Section 3.2, we define the structure of a *Blueprint* with aim to support the development of the BSL and the BMTs afterwards.
- In Section 3.3, we introduce the BSL as the first component of the *Blueprint Approach*.
- In Section 3.4, we introduce the BMTs as the second component of the *Blueprint Approach*.
- In Section 3.5, we explain how the BSL and BMTs could be used by cloud service providers and CSBA engineers within the htCSBA engineering lifecycle.

3.1 Introduction

The *Blueprint Approach* has been developed to support the design and configuration of an CSBA with the two components: the Blueprint Specification Language (BSL) and the Blueprint Manipulation Technique(BMTs). In particular:

- The *Blueprint Specification Language* (BSL): provides a specification language for cloud service providers to abstractly (i.e., independent of implementation) and unambiguously *specify* their cloud services on any layer of the cloud stack, i.e., SaaS, PaaS or IaaS. Figure 3.1 shows how cloud services can be specified in a series of SaaS, PaaS, and IaaS blueprints using the BSL. Please note that we consider an CSBA as a composite SaaS and hence use the SaaS blueprint for its specification. The structure of a blueprint will be defined in the next Section 3.2.1, which includes the definitions of the fundamental elements and the dependency links of a blueprint.

- The *Blueprint Manipulation Techniques* (BMTs): provide a set of manipulation techniques that operate on the blueprints. These techniques allow a cloud service provider to *publish* his blueprints to a marketplace repository or *delete* his existing blueprints in the repository. The BMT also supports an CSBA engineer to *query* for the needed blueprints, and to flexibly *compose* multiple blueprints to configure his CSBA. Figure 3.2 illustrates these BMT techniques.

  Composing blueprints results in creating the dependency links between them. The precise definition of a dependency link between blueprints will be given in the next Section 3.2.2. Such a dependency link can exist between two blueprints on the same or two adjacent layers. A blueprint may also have two alternative dependency links, e.g. the `SaaS-2` blueprint in Figure 3.2 has two alternative dependency links on the `PaaS-1` and `PaaS-2` blueprints.

Figure 3.1: Blueprint Approach - *Blueprint Specification Language*

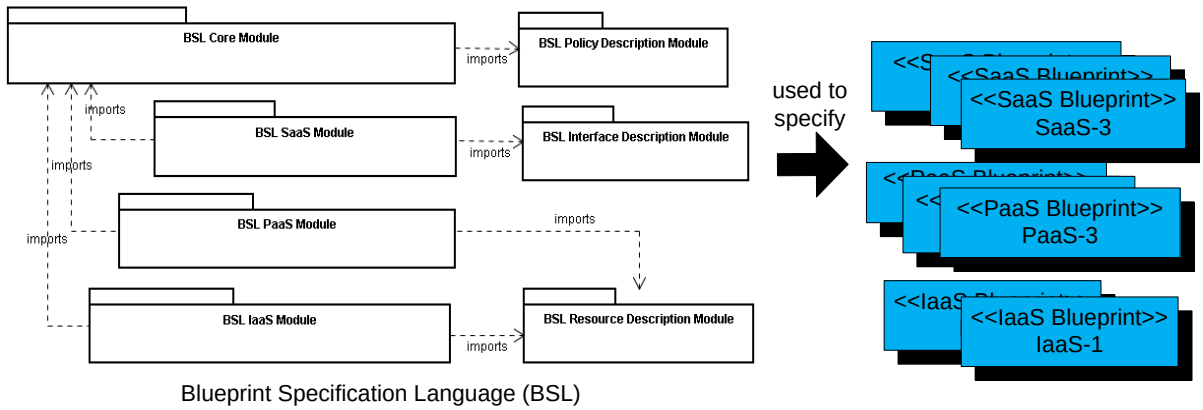
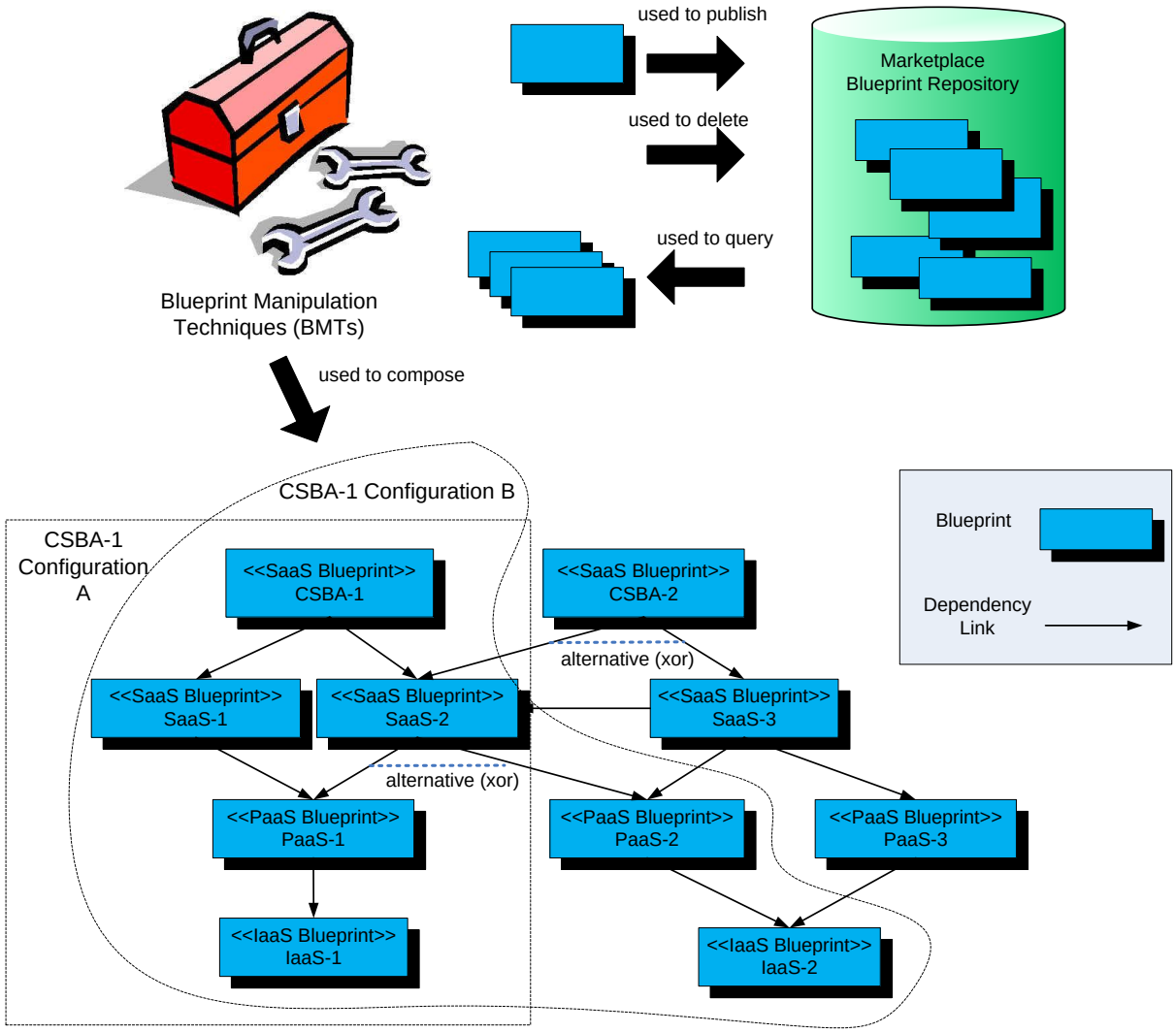


Figure 3.2: Blueprint Approach - *Blueprint Manipulation Techniques*



Together, the *BSL* and *BMTs* provide a comprehensive method for (1) creating cloud service and CSBA specification using blueprints, (2) publishing blueprints to a blueprint repository, (3) deleting existing blueprints in a blueprint repository, (4) querying blueprints from a blueprint repository, and (5) composing blueprints to configure an CSBA. In Figure 3.2, by composing multiple blueprints, two alternative configurations for the `CSBA-1` can be created, namely the `CSBA-1-Configuration-A` and `CSBA-1-Configuration-B`. The reason of having two alternative configurations for the `CSBA-1` is because the blueprint `SaaS-2` has two alternative dependency links on the `PaaS-1` and `PaaS-2` blueprints.

Having explained the use of the *BSL* and *BMTs* for designing and configuring an CSBA, we claim that using the *Blueprint Approach* delivers the following benefits for an CSBA engineer:

- The *Blueprint Approach* aims to support a full automation and optimization of

CSBA configuration by allowing cloud service specification at any layer of the cloud stack to be appropriately composed with (through the dependency links) or decomposed from other cloud service specifications at the same or on another adjacent layer. It also seeks to simplify the complexity of managing all alternative configurations of an CSBA with a variety of third-party SaaS, PaaS and IaaS options. As an example in Figure 3.2, all possible configurations of the `CSBA-1` can be derived and managed efficiently. Based on some predefined criteria, an optimum configuration of the `CSBA-1` can be easily determined.

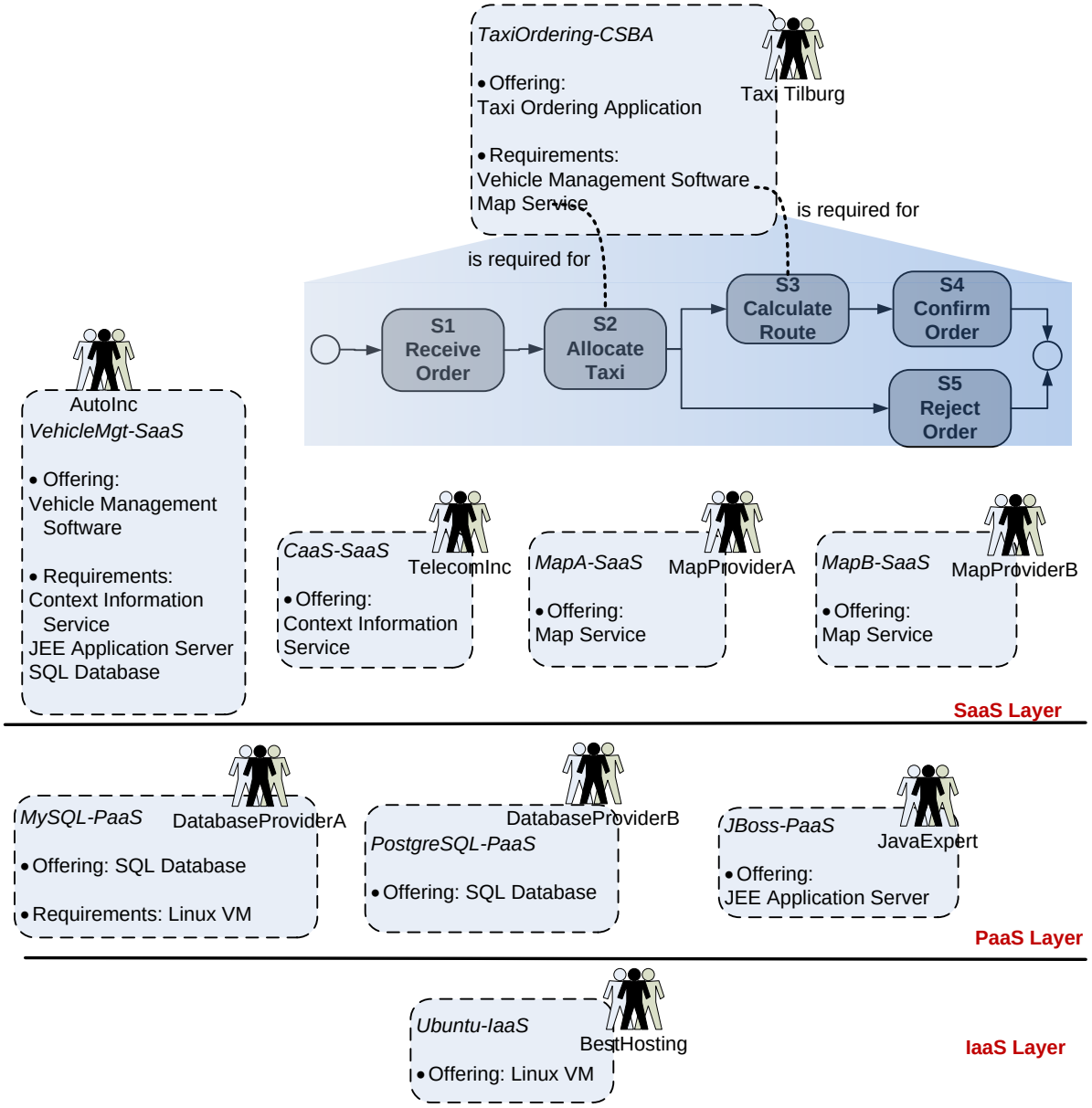
- Portability of an SaaS across several PaaS/IaaS clouds can also be derived to leverage the benefits of elasticity and scale. As an example, from the CSBA configuration in Figure 3.2, it can be derived that there are two alternative deployment options for the `SaaS-2`: the `PaaS-1` and `PaaS-2`. In other words, `SaaS-2` is portable between `PaaS-1` and `PaaS-2`.

- Traceability can also be supported for configuration decisions throughout all top-down cloud layers, from the application layer down to the platform and infrastructure layers or vice versa. This results into an ideal closed-feedback loop for selecting an optimum configuration or improving an existing configuration of an CSBA. As an example, the `CSBA-1` in Figure 3.2 may be tested with the two alternative configuration `CSBA-1-Configuration-A` and `CSBA-1-Configuration-B` and the test yields that the performance of the `CSBA-1-Configuration-A` configuration is better than the performance of the `CSBA-1-Configuration-B` configuration. Based on the performance indicators exposed in the `PaaS-1` and `PaaS-2` blueprints, an CSBA engineer is able to discover the reason that hosting the `SaaS-2` on the `PaaS-2` results in a better performance than hosting it on the `PaaS-1`.

From the architecture point of view, the *Blueprint Approach* supports the independent layering of cloud services within a typical cloud stack, i.e. the cloud services can now be considered as composable and interchangeable building blocks of an CSBA configuration. Hence, the *Blueprint Approach* promotes the two fundamental characteristics of a SOA: the reuse and composition of independent, loosely-coupled cloud services across three layers of the cloud stack. Practically speaking, by adopting the *Blueprint Approach*, an CSBA engineer can now freely pick and choose discrete SaaSs from multiple providers to compose a coherent and integrated CSBA, and for each of these constituting SaaSs there is a variety of alternative PaaS/IaaS solutions for its deployment and provisioning.

Prior to the development of the BSL and the BMT, it is necessary to define the structure of a blueprint, i.e. the fundamental elements of a blueprint, and the dependency links between blueprints. The next Section 3.2 discusses this topic. Then, Section 3.3

Figure 3.3: Actors and their Cloud Services in the *Taxi Tilburg Scenario*.



and Section 3.4 provide more details about the two components of the *Blueprint Approach*, i.e. the BSL and the BMTs respectively. Finally, Section 3.5 explains how the BSL and BMTs can be used to support the cloud service providers and CSBA engineers within the CSBA engineering lifecycle.

Case Study: the *Taxi Tilburg Scenario*

To motivate the use of the *Blueprint Approach* within the CSBA engineering lifecycle, we introduce in this section a scenario called **Taxi Tilburg Scenario**, which is about the design and configuration of an CSBA in the today's industrial reality. The scenario is a modification of the *Taxi Application Scenario* that was originally co-developed with several industrial IT companies like Telefonica, Telecom Italia, Ericson, 2ndQuadrant and SAP as a case study for the EC's 4CaaSt project [European Comission, 2010]. In this scenario, a company called `TaxiTilburg` would like to develop an CSBA that can support the tasks of receiving an order for taxi from a customer, checking current status of taxi fleet to allocate a taxi, calculating the route to the customer, and finally, confirming or rejecting the order. `TaxiTilburg` decides to follow the CSBA engineering lifecycle to design and configure their CSBA by assembling cloud service specifications that can be retrieved from a cloud service marketplace.

Figure 3.3 presents the actors involved in the scenario that collaborate in a marketplace, through which their cloud services can be advertised and purchased. The actors and their cloud services are briefly explained as follows.

BestHosting: is an IaaS provider that provides different types of Virtual Machines (VMs). One of his IaaSs is the *Ubuntu-IaaS* that offers "Linux Virtual Machines (VMs)". Each VM is configured with 3Ghz CPU speed and 3 GiB memory size.

`JavaExpert`: is a PaaS provider, specialized in hosting Java based applications. The provider offers a PaaS called *JBoss-PaaS*, which is a pre-configured "JEE Application Server" running on his on-premise infrastructure. Each customer can request up to 4 instances of the *JBoss-PaaS*. The *JBoss-PaaS* is elastic as its computing resource capacity can be scaled up and down in terms of CPU speed (Ghz) and memory size (GiB). More specifically, the *JBoss-PaaS* can be configured to have the CPU speed between 2 Ghz and 4 Ghz and the memory size between 2 GiB and 4 GiB. Besides, its network link can have a bandwidth between 2 and 3 Gbit/s.

The other PaaS providers in the scenario are `DatabaseProviderA` and `DatabaseProviderB`. They are experts in providing storage services, especially SQL databases. The difference in their "SQL database" offerings is that `DatabaseProviderA` allows their customers to request for maximum 4 instances of their *MySQL-PaaS*, each with 3 TB storage, whilst `DatabaseProviderB` supports only up to 3 instances of *PostgreSQL-PaaS*, each with 2 TB storage. Furthermore, whilst `DatabaseProviderB` always hosts his *PostgreSQL-PaaS* in-house, `DatabaseProviderA` allows their customers to deploy his *MySQL-PaaS* on an external "Linux VM".

On the SaaS layer, `Auto Inc.` is an established medium-scale enterprise that has spotted a business opportunity providing a "Vehicle Management Software" in the Netherlands. They plan to offer the software as a SaaS called *VehicleMgt-SaaS*, since this provides ubiquitous and common access for their prospective customers, e.g., taxi providers and car-hiring providers. To implement the *VehicleMgt-SaaS*, AutoInc has contracted a software company to develop a vehicle management software in Java. The software contains a number of binary files, e.g. the .war and .jar files, and data configuration files, e.g. database dump file. These artefacts require a "JEE Application Server" and a "SQL database" for their deployment. To obtain the context information of each of the vehicle in the fleet, AutoInc relies on external "Context Information Service" provider for this functionality. Regarding the Quality of Service (QoS) of their *VehicleMgt-SaaS*, `Auto Inc.` promises to their customers that the maximum response time of their SaaS is 6s and the minimum availability of their SaaS is 98% on 24/7. To meet this objective, experts of `Auto Inc.` have determined that they need to deploy the vehicle management SaaS on an JEE application server with CPU Speed $\geq$ 2Ghz and memory size $\geq$ 2 GiB, and they need a SQL database with capacity $\geq$ 2 TB.

The other SaaS providers include: the `TelecomInc` who provides a "Context Information Service" called *CaaS-SaaS*, the `MapProviderA` who provides a "Map Service" called *MapA-SaaS*, and the `MapProviderB` who provides another "Map Service" called *MapB-SaaS*. All these three SaaSs are hosted on-premise. There is no information regarding the QoS of the *CaaS-SaaS*. In contrast, the `MapProviderA` promises that their *MapA-SaaS* has a response time $\leq$ 3s and the availability $\geq$ 98%. A slightly better QoS is offered by the `MapProviderB` for their *MapB-SaaS*, i.e. response time $\leq$ 2s and the availability $\geq$ 99%.

`TaxiTilburg` is an CSBA provider that offers the "Taxi Ordering Application" in the form of an end-to-end process-based CSBA called *TaxiOrdering-CSBA*, as shown in Figure 3.3. The process is composed of several steps including Receive Order, Allocate Taxi, Calculate Route, Confirm Order, and Reject Order. In order to implement the steps S1 (Receive Order), S4 (Confirm Order), S5 (Reject Order), TaxiTilburg has developed an in-house SMS software component that is able to receive new orders by sms, and send order confirmations or rejections to customers also by sms. Unfortunately, due to lack of expertise, TaxiTilburg has to rely on third-party SaaS offerings for managing its taxi fleet and for calculating routes, which are the two necessary functionalities to implement step S2 (Allocate Taxi) and step 3 (Calculate Route) in its TaxiOrdering-CSBA process. These two required functionalities are captured as the two functional requirements "Vehicle Management Software" and "Map Service".

TaxiTilburg has promised to its customers the QoS level $(responseTime(s) \leq 15)\&(availability(\%) \leq 98)$ of their CSBA. To maintain this predefined QoS level, TaxiTilburg has distributed the required maximum response time to each step contained in TaxiOrdering-CSBA process. For the step S2 (Allocate Taxi) that relates to the requirement "Vehicle Management Software", the required maximum response time is set to 6s, and for the step 3 (Calculate Route) that relates to the requirement "Map Service", it is set to 3s. The required minimum avaialability 98% remains same for all steps.

Table 3.1 summarizes all the cloud services in the Taxi Tilburg Scenario including (1) their offerings and requirements, and (2) the QoS and resource specifications for their offerings and requirements.

The **Taxi Tilburg Scenario** promotes a uniform specification language to exist so that cloud services across all three layers can be specified in a uniform way. This language should also allow to specify the policy in general and QoS in particular of a cloud service. Resource specification for PaaSs/IaaSs should also be supported by this language. Furthermore, the scenario also motivates the need of manipulation techniques that supports the publishing, querying, modifying, and composing cloud service specifications to configure an CSBA. As an example, cloud service providers in this scenario have a desire to publish their cloud service specifications to a marketplace repository. Some cloud service providers, e.g. the `Taxi Tilburg` and `AutoInc`, would like to query the marketplace repository to retrieve cloud services that can match their requirements. In particular, `Taxi Tilburg` is the one who would like to compose external cloud service specifications to configure their *Taxi Ordering CSBA*.

Throughout the rest of the thesis, this scenario will also be used as a running example to illustrate how the *Blueprint Approach* can be applied to support a uniform cloud service specification language and a set of cloud service manipulation techniques for configuring an CSBA.

Table 3.1: QoS and Resource Specifications for Cloud Services in the *Taxi Tilburg Scenario*.

Offering or Requirement	QoS Specification	Resource Specification
Cloud Service - ***TaxiOrdering-CSBA***		
Offering - `Taxi Ordering Application`	(responseTime(s) $\leq$ 15) & (availability(%) $\geq$ 98)	n/A
Requirement - `Vehicle Management Software`	(responseTime(s) $\leq$ 6) & (availability(%) $\geq$ 98)	n/A
Requirement -`Map Service`	(responseTime(s) $\leq$ 3) & (availability(%) $\geq$ 98)	n/A
Cloud Service - ***MapA-SaaS***		
Offering - `Map Service`	(responseTime(s) $\leq$ 3) & (availability(%) $\geq$ 98)	n/A
Cloud Service - ***MapB-SaaS***		
Offering - `Map Service`	(responseTime(s) $\leq$ 2) & (availability(%) $\geq$ 99)	n/A
Cloud Service - ***VehicleMgt-SaaS***		
Offering - `Vehicle Management Software`	(responseTime(s) $\leq$ 6) & (availability(%) $\geq$ 98)	n/A
Requirement -`Context Information Service`	n/A	n/A
Requirement -`SQL database`	n/A	capacity(TB) $\geq$ 2
Requirement -`JEE Application Server`	n/A	(CPUSpeed(Ghz) $\geq$ 2) & (memory(GiB) $\geq$ 2) & (networkBandwidth(GBit/s) $\geq$ 2)
Cloud Service - ***CaaS-SaaS***		
Offering - `Context Information Service`	n/A	n/A
Cloud Service - ***Jboss-PaaS***		
Offering - `JEE Application Server`	n/A	(2 $\leq$ CPUSpeed(Ghz) $\leq$ 4) & (2 $\leq$ memory(GiB) $\leq$ 4) & (2 $\leq$ networkBandwidth(Gbit/s $\leq$ 3)
Cloud Service - ***PostgreSQL-PaaS***		
Offering - `SQL database`	n/A	capacity(TB) = 2
Cloud Service - ***MySQL-PaaS***		
Offering - `SQL database`	n/A	capacity(TB) = 3
Requirement -`Linux VM`	n/A	n/A
Cloud Service - ***Ubuntu-IaaS***		
Offering - `Linux VM`	n/A	(CPUSpeed(Ghz) = 3) & (memory(GiB) = 3)

3.2 Blueprint Structure Definition

In this Section, we define the structure of a blueprint. This task is necessary for the development of the BSL and BMTs as it provides a precise understanding of what types of information are captured in a blueprint.

The concept of a blueprint has been defined in Definition 1.4 as a uniform, implementation-agnostic cloud service specification on any layer of the cloud stack. Definition 1.4 has also identified the information sets that should be contained in a blueprint. To enrich this definition, we also aimed to derive the requirements from a group of experts in the cloud computing domain. The process of eliciting requirements was rather informal, i.e. we have only organized project meetings and online workshops among the 4caast community [European Comission, 2010] to understand their requirements for a common, standardized cloud service specification. The participants in these workshops and meetings comprise of both academic and industry partners. As a result, we were able to enrich the Definition 1.4 with the desired features that are relevant to the practical needs of current industrial cloud providers.

The blueprint structure introduced in this section is defined based on the information sets identified in Definition 1.4. It has also been consolidated with the desired features proposed by our industrial partners. We present in Definition 3.1 the derived blueprint structure.

Definition 3.1 (Blueprint Structure) *A blueprint comprises of:*

- *A set of fundamental blueprint elements.*
- *A set of dependency links between the blueprint elements.*
- *A set of policy profiles and resource profiles that are attached to blueprint elements to specify their policy and resource properties.*

The rest of this section refines the Definition 3.1 through the following topics:

- Section 3.2.1 defines the fundamental *blueprint elements*.
- Section 3.2.2 defines the *dependency links* between the blueprint elements.
- Section 3.2.3 defines a *policy profile*.
- Section 3.2.4 defines a *resource profile*.
- Section 3.2.5 introduces the *blueprint classification criteria* based on the blueprint structure introduced in this section.

It is worth to mention that within the EC's 4caaSt FP7 project [European Comission, 2010] that involves industrial key players in cloud computing such as Telefonica, SAP, Erricson, etc., the derived blueprint structure has been continuously used as a standard, uniform and implementation-agnostic specification of a cloud services. It has also been validated that the blueprints defined following our structure definition are capable to capture all the necessary aspects of an industrial cloud service, yet still remain simple enough to be used by our industry partners.

3.2.1 Blueprint Elements

Definition 3.2 defines the fundamental elements of a blueprint.

Definition 3.2 (Blueprint Elements) *A blueprint includes the following elements:*

- *An Offering: specifies the functional and non-functional characteristics of the offered cloud service.*
- *Zero or many Deployment Artifacts: are parts of the offering. They are the physical pieces of information that need to be deployed or installed to enable the offering.*
- *Zero or many Requirements: specify the functional and non-functional characteristics of the required cloud services.*

In the following, we will explain the details of these three blueprint elements.

Offering

An offering represents the cloud service offered by the blueprint. It specifies the *functional and non-functional characteristics* of the offered cloud service. The offerings on different layers of the cloud stack may expose different characteristics. Our assumption of classifying cloud services based on the three layers of the cloud stack results into the classification of three different types of blueprints and offerings: (1) a SaaS blueprint contains a SaaS offering, (2) a PaaS blueprint contains a PaaS offering, and (2) an IaaS blueprint contains an IaaS offering. In fact, most of the current commercial cloud service offerings can be easily classified into these three types, e.g. Amazon EC2, Flexiant, VMWare, GoGrid are the most known IaaS offerings; Microsoft Azures, Force.com, and Google App Engine are typical PaaS offerings; and SalesForce, Dropbox and Google App are the typical SaaS offerings.

The *general, layer-agnostic functional characteristics* of an offering include:

- Functionality: indicates what is offered as a cloud service to the consumers. Examples of functionalities may cross all three layers of the cloud stack and include

for instance: "SMS service" (SaaS), "Map Service" (SaaS), "Servlet Container" (PaaS), "SQL Database" (PaaS), "Linux Virtual Machine" (IaaS), "Ethernet Network Link" (IaaS), etc.

- Range of instances: indicates the minimum and maximum number of cloud service instances that can be provided to a single customer.
- Multi-tenancy: indicates whether a single cloud service instance can simultaneously serve several customers.
- Endpoint (optional): indicates the location where the cloud service can be accessed. This characteristic is specified only in the case the cloud service has already been deployed and is ready to consume.

On each specific layer of the cloud stack, the offering may expose further *layer-specific functional characteristics* including:

- On the SaaS layer: the *technical interface* specifies the signature and protocol of a SaaS. Its purpose is to enable the programmatic interaction between the consumers and the SaaS. As an example, the technical interface of a SaaS may include a WSDL document as its signature and a WS-BPEL document as its protocol.
- On the PaaS and IaaS layers:
  - Information about the *product and technology* used to implement a PaaS/IaaS: This characteristic provides some transparancy for the consumers to deploy their applications. Examples of products and technologies include the "JBoss Application Server" product that implements both the "Servlet Container" and the "JEE Application Server" technologies.
  - The resource description of a PaaS/IaaS: indicates the computing, storage and network resource capacity that is currently being provisioned to the PaaS/IaaS to function properly.

Regarding the *non-functional characteristics* of an offering, we specify them in terms of the policy characteristics regarding the communication with the service, service selection, and service usage. Policy characteristics can be specified for an offering on any layer of the cloud stack. The topic of policy specification in a blueprint will be followed up in the next Section 3.2.3.

Deployment Artifact

Inspired by the definition of an UML artifact[1], a deployment artifact in a blueprint is the specification of a physical piece of information that is part of an offering and

[1]OMG, OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2, p197

normally used for deploying or installing the offering. The specification of a deployment artifact includes its name, its type (binary file, configuration file, data dump file, composition script, etc.) and the location to retrieve it.

On the SaaS layer, examples of deployment artifacts of a SaaS offering include the binary files or deployable application packages that implement the SaaS, or the configuration files for configuring its deployment environment. On the PaaS layer, platform installation binaries, e.g. Tomcat .jar binaries, are the essential deployment artifacts of a PaaS offering. On the IaaS layer, the virtual images and/or virtual network configuration files, e.g. an OVF [DMTF, b] or AMI[2], are examples of deployment artifacts of an IaaS offering.

Requirement

A requirement specifies a required cloud service in the blueprint. It could be a SaaS, PaaS, or IaaS requirement indicating a required SaaS, PaaS, or IaaS provided by a third-party cloud vendor. A requirement and an offering on the same layer of the cloud stack have the similar structure, since both of them specify an abstract cloud service that is either offered or required by the blueprint. The only difference is that a requirement cannot be specified with an endpoint whilst an offering, once it has been deployed, can be specified with a concrete endpoint.

A requirement can also be attached with a policy specification to indicate its requested policy capabilities, or a PaaS/IaaS requirement can also be attached with a resource specification to specify its requested resource capabilities.

Examples of Blueprints and Blueprint Elements in the *Taxi Tilburg Scenario*

Each blueprint and its elements are assigned with a unique ID. Figure 3.4 introduces four blueprints: (1) the blueprint with the unique ID `TaxiOrdering-CSBA-BP` is the specification of the *TaxiOrdering-CSBA* that supports an CSBA for ordering a taxi, (2) the blueprint with the unique ID `VehicleMgt-BP` is the specification of the *Vehiclemgt-SaaS* that provides a software for managing vehicle fleets, (3) the blueprint with the unique ID `MySQL-BP` is the specification of the *MySQL-PaaS* that offers a MySQL database server, and (4) the blueprint with the unique ID `Ubuntu-BP` is the specification of the *Ubuntu-IaaS* that offers a Linux virtual machine. Let us focus the sample blueprint `VehicleMgt-BP` in Figure 3.4 and explain its elements in more details:

- `VehicleMgt-BP` is a SaaS blueprint that provides a SaaS offering (unique ID=`VehicleMgt-Off`) for managing vehicle fleets.
- Two deployment artifacts (unique ID=`App.war`) and (unique ID=`Data.zip`) exist as parts of the `VehicleMgt-Off` offering. The `App.war` is a war file containing the entire vehicle management

[2]Amazon Machine Images: https://aws.amazon.com/amis

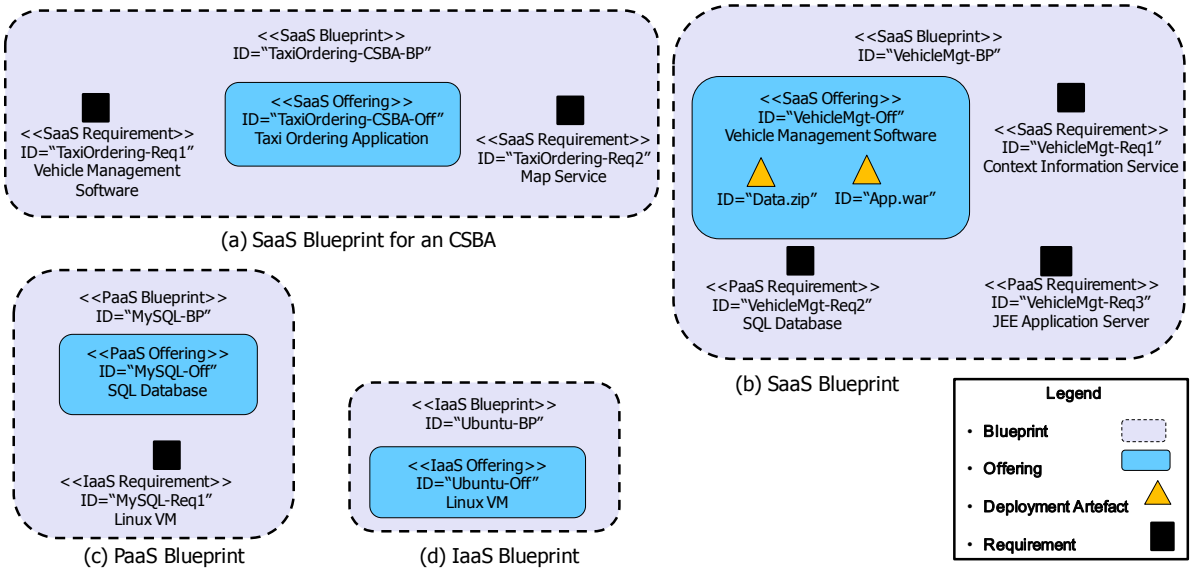
application binaries. The `Data.zip` contains all the data dump files and data configuration scripts to be installed in a relational database.

- The `VehicleMgt-BP` requires the following third-party cloud services:
  - A context information service (unique ID=`VehicleMgt-Req1`) is required to provide the surrounding context information of each vehicle. It is classified as a SaaS requirement in the `VehicleMgt-BP` blueprint.
  - A relational database (unique ID=`VehicleMgt-Req2`) is required for the installation of the deployment artifact `Data.zip` so that all the needed data of the vehicle management application can be correctly configured and persisted. It is classified as a PaaS requirement in the `VehicleMgt-BP` blueprint.
  - A JEE application server (unique ID=`VehicleMgt-Req3`) is required for the deployment of the deployment artifact `App.war`. It is classified as a PaaS requirement in the `VehicleMgt-BP` blueprint.

Although we did not specify an IaaS requirement for an IaaS blueprint in our case study, this case in general is still valid. One of the most common examples is that an IaaS blueprint requires a network link, which is classified as an IaaS requirment in our definition. For instance, our case study could be extended so that the `Ubuntu-BP` blueprint can contain an IaaS requirement for a "Ethernet 2Gbit network link".

The definition of the fundamental elements of a blueprint (Definition 3.2) makes it become a self-contained, composable units with an offering and a set of requirements. Configuring end-to-end CSBAs using blueprints follows the principles in component-based software engineering [Szyperski, 2002] and service compo-

Figure 3.4: Examples of Blueprints and Blueprint Elements in the *Taxi Tilburg Scenario*



sition [Papazoglou, 2003] to compose a number of discrete blueprints by (1) matching the requirements of a blueprint with the offerings of other existing blueprints, and (2) linking the requirements with the matched offerings through the "dependency links". Whilst the support for matching and linking between an offering and a requirement is part of the Blueprint Manipulation Technique (BMT) introduced in the subsequent section 3.4, we define in the following Section 3.2.2 the "dependency links" that may exist within a blueprint and between two blueprints.

3.2.2 Blueprint Dependency Links

Former related work in component-based system formally defines the two forms of dependency links for a component-based software deployment: the the intra- and inter-dependency links [Belguidoum & Dagnat, 2007]. Intra-dependency links exist between the provided and required services of the same component. They are defined by the component producer and used to perform the installation. Inter-dependency links result from the installation and indicate the dependencies between the required services of a component with the provided services of other components. They are typically used to perform deinstallation and replacement.

Inspired by this definition of component dependency links, one of the options to define the dependency links between the blueprint elements is to follow the same concepts of the component dependency links. In particular, we could have two types of dependency links: (1) the internal ones between an offering and the requirements of the same blueprint, and (2) the external ones between a requirement and an offering of two distinct blueprints. However, one of the drawback of this solution is not to able to distinguish between the dependency between two elements on the same cloud layer (normally the functional dependency) and the dependency between two elements on two adjacent layers (normally the deployment dependency). Given this reason, we decided to take into account the cloud layers and define in the Definition 3.3 the two types of blueprint dependency links: the *vertical* and *horizontal* links. Then in Proposition 3.1, we restrict the existence of these two types of blueprint dependency links. We will also explain afterwards the assumptions that have resulted into the restriction rules.

Definition 3.3 (Blueprint Dependency Links) *A blueprint dependency link indicates a functional dependency between two blueprint elements. It has one of the following two types:*

- ***Vertical Link**: exists between two blueprint elements on two adjacent layers of the cloud stack. The blueprint element on the higher layer is the depender and the blueprint element on the lower layer is the dependeee. For instance, a SaaS offering can have a vertical link on a PaaS requirement.*
- ***Horizontal Link**: exists between two blueprint elements on the same layer of the cloud stack. For instance, a SaaS offering can have a horizontal link on a SaaS requirement.*

Proposition 3.1 (Restrictions on the Blueprint Dependency Links) *We define the following restrictions on the existence of a vertical or horizontal link:*

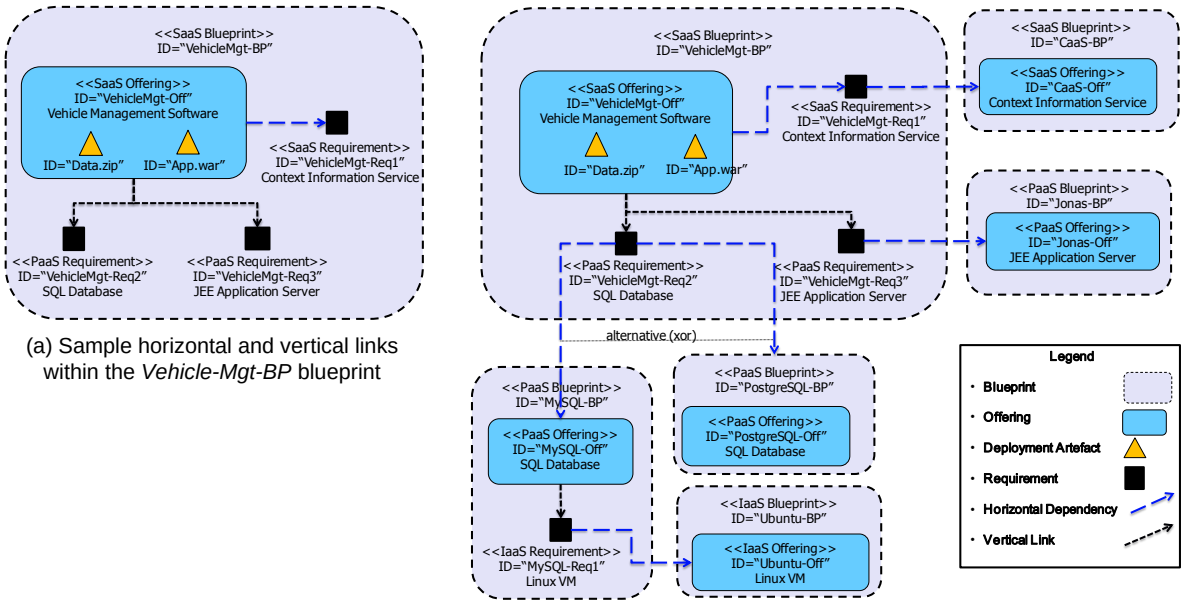
- *A Vertical Link: can only exist between an Offering (the depender on the higher cloud layer) and a Requirement (the dependee on the lower cloud layer) within the same blueprint.*
- *Horizontal Link: can only exist*
  - *between an Offering (the depender) and a Requirement (the dependee) within the same blueprint.*
  - *between a Requirement (the depender) and an Offering (the dependee) of two different blueprints.*

The restrictions of blueprint dependency links have been derived due to the following assumptions:

1. *A Deployment Artefact does NOT have any dependency links*: A deployment artefact is part of an offering and thus does not need to have any explicit dependency links. It is normally associated with an installation/deployment cookbook containing the instructions of which requirements (of the blueprint) are needed for its deployment and how to perform the deployment, e.g. using the Chef cookbook for deploying Tomcat[3].
2. *A Requirement-to-Requirement dependency link does NOT exist*: Our assumption states that if a requirement has a dependency link on another requirement, these two should be defined together in a single requirement.
3. *An Offering-to-Offering dependency link does NOT exist*: Our assumption states that

[3] `http://community.opscode.com/cookbooks/tomcat`

Figure 3.5: Examples of Vertical and Horizontal Links in the *Taxi Tilburg Scenario*



(a) Sample horizontal and vertical links within the *Vehicle-Mgt-BP* blueprint

(b) Sample horizontal links among several blueprints

this type of dependency link can always be represented through a composition of an Offering-to-Requirement dependency link and a Requirement-to-Offering dependency link.

4. *An Offering-to-Requirement dependency link does NOT exist between two blueprints*: This assumption is due to the fact that we would like to keep a blueprint as a self-contained, composable unit so that its offering can only depends on internal requirements. Furthermore, an external requirement can always be replicated to an internal requirement of a blueprint.

5. *A Requirement-to-Offering dependency link does NOT exist within a blueprint*: A requirement represents an external cloud service required by the blueprint and obviously cannot have a dependency link on the offering of the same blueprint.

6. *A Requirement-to-Offering dependency link between two blueprints does NOT exist as a vertical link*. A Requirement-to-Offering dependency link between two blueprints indicates that the requirement of a blueprint can be fulfilled by the offering of another blueprint. We assume that a requirement can only be fulfill by an offering on the same cloud layer. Hence, this dependency link is always a horizontal link.

Examples of Vertical and Horizontal Links in the *Taxi Tilburg Scenario*

Following up the example in Figure 3.4, the sample vertical and horizontal links within the `VehicleMgt-BP` blueprint are introduced in Figure 3.5(a) and yield the following information:

- The SaaS offering `VehicleMgt-Off` wants to reuse the context information service functionality of a third-party SaaS (represented by the SaaS requirement `VehicleMgt-Req1`). There is a horizontal link between `VehicleMgt-Off` and `VehicleMgt-Req1` to indicate this dependency.
- The SaaS Offering `VehicleMgt-Off` requires a SQL database (represented by the PaaS requirement `VehicleMgt-Req2`) to deploy the artifact `Data.zip`. There is a vertical link between `VehicleMgt-Off` and `VehicleMgt-Req2` to indicate this dependency.
- The SaaS offering `VehicleMgt-Off` requires a JEE application server (represented by the PaaS requirement `VehicleMgt-Req3`) to deploy the artifact `App.war`. There is a vertical link between `VehicleMgt-Off` and `VehicleMgt-Req3` to indicate this dependency.

Figure 3.5(a) shows that the `VehicleMgt-BP` blueprint is still "unresolved", i.e. it still contains three unmatched requirements. In the further configuration activities for the `VehicleMgt-BP` blueprint, the three requirements need to be matched by querying, selecting and composing appropriate cloud service offerings of other blueprints that can be retrieved from the marketplace repository. Matching a requirement with an offering creates a horizontal link between them. The `VehicleMgt-BP` blueprint becomes "resolved" if all of its requirements have horizontal links with the offerings of the other blueprints.

Figure 3.5(b) visualizes the sample "resolved" `VehicleMgt-BP` together with the other blueprints in the Taxi Tilburg Scenario. The requirement `VehicleMgt-Req1` now has a horizontal link with the `CaaS-Off` offering of the `CaaS-BP` blueprint, the requirement `VehicleMgt-Req3` has a horizontal link with the `Jonas-Off` offering of the `Jonas-BP` blueprint, and the requirement `VehicleMgt-Req2` has two alternative horizontal links, one with the `MySQL-Off` offering of the `MySQL-BP` blueprint and the other one with the `PostgreSQL-Off` offering of the `PostgreSQL-BP` blueprint.

3.2.3 Policy Profiles in a Blueprint

In the previous Section 3.2.1, the offering and requirement have been defined as the two essential elements of a blueprint. An offering has been defined as the provided cloud service of a blueprint and a requirement has been defined as a required, external cloud service. In the context of cloud computing where an external cloud service offered by a third-party provider is certainly not under control of the users, it is of

particular importance to consider its *policy properties*, e.g. Quality of Service (QoS)[4], security, privacy, cost, etc, before actually selecting and using it. In this section, we define the concept of a *Policy Profile* that can be attached to an offering or requirement to specify the assertions on its policy properties.

Policy properties of an offering or requirement can be specified in the form of *policy assertion* following the definition in the WS-Policy framework [W3C, 2006]: "A policy assertion represents a requirement, capability, or other property of a behavior ... Some policy assertions specify traditional requirements and capabilities that will manifest themselves in the messages exchanged (e.g., authentication scheme, transport protocol selection). Other policy assertions have no wire manifestation in the messages exchanged, yet are relevant to service selection and usage (e.g., privacy policy, QoS characteristics)". Policy assertions according to this definition cover a vast area of policy properties, i.e. from the ones related to message exchange to the ones related to service selection and usage, for instance QoS and privacy properties. If following the WS-Policy framework, policy assertions can be further grouped into different *policy alternatives* within a policy profile. For brevity reason we assume that a policy profile does not need to be defined with different policy alternatives. Definition 3.4 defines a policy profile.

Definition 3.4 (Policy Profile) *A policy profile can be specified for an offering or requirement of a blueprint. It is a non-ordered collection of policy assertions. A policy assertion specifies the value range for a policy property.*

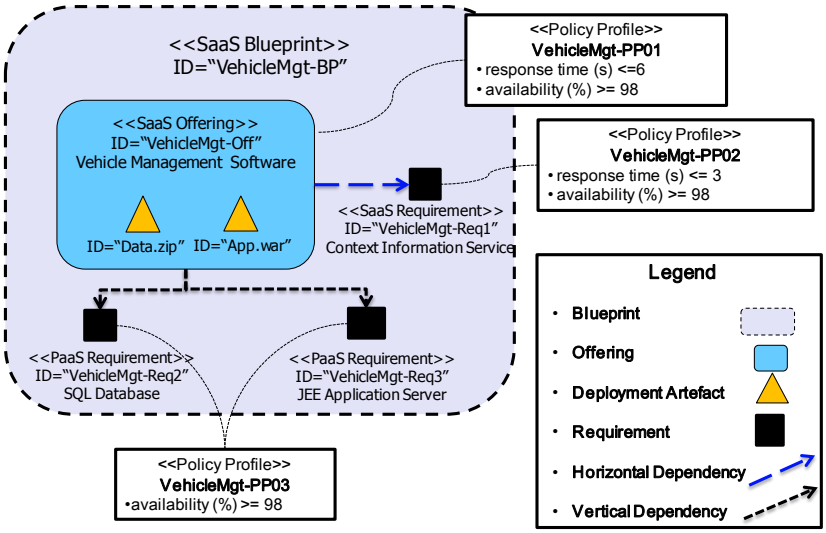
Unfortunately, the WS-Policy framework does not come with a concrete taxonomy of policy properties for (Web) services. It would rather leave the definition of policy properties to users' hand.

The S-Cube Quality Reference Model (QRM) presented in [Gehlert & Metzger, 2008] has been developed by the S-Cube consortium[5] and is intended to provide a unified terminology for describing different QoS properties of SBAs. Quality models from diverse domains such as service-oriented-computing, business process management, grid computing, and software engineering have been analyzed in order to extract the relevant QoS properties for SBAs. The QoS properties in the QRM are organized into categories such as performance, dependability, usability, etc., and provides a clear hierarchy between the QoS categories and properties, e.g. performance contains response time and throughput and latency is a type of response time. Furthermore according to their definition, some of the QoS categories cover also other policy characteristics of a cloud service such as security, cost, and data policy. Hence, the S-Cube QRM seems to be an appropriate reference point for

[4] `http://www.s-cube-network.eu/km/terms/q/quality-of-service-qos`

[5] S-Cube Network of Excellence: `http://www.s-cube-network.eu/`

Figure 3.6: Example of Policy Profiles in the *Taxi Tilburg Scenario*



defining policy properties for a cloud service.

To exemplify the use of policy properties in our work, we select from the S-Cube QRM the two policy properties *Response Time (in s)* and *Availability (in %)*. These policy properties have been selected based on our observation on the most concerned policy properties of current commercial cloud provider, e.g. Amazon. With these selected policy properties, policy assertions can be specified as the value range of the policy properties (with a maximum value, or a minimum value, or both), e.g. "$ResponseTime(s) \leq 6$" or "$Availability(\%) \leq 98$".

Examples of Policy Profiles in the *Taxi Tilburg Scenario*

Examples of policy profiles of the sample `VehicleMgt-BP` blueprint are given in Figure 3.6. The policy profile `VehicleMgt-PP01` specifies the assertions on the response time and availability of the `VehicleMgt-Off` offering, i.e. $ResponseTime(s) \leq 6$ & $Availability(\%) \geq 98$. To enable these policy assertions, the provider of the `VehicleMgt-BP` blueprint has specified two further policy profiles: (1) `VehicleMgt-PP02` that contains the assertions on the required response time and availability of the SaaS requirement `VehicleMgt-Req1`, and (2) the policy profile `VehicleMgt-PP02` that contains the assertion on the required availability of the two PaaS requirements `VehicleMgt-Req2` and `VehicleMgt-Req3`.

3.2.4 Resource Profiles in a Blueprint

As discussed in the previous Section 3.2.3, policy properties specify the QoS, security, privacy, cost, etc, with aim to support the users with the selection of third-party cloud

services. In case the users intend to use a cloud service as it is and are not much interested in configuring its underlying infrastructure, policy properties are the key supportive non-functional properties during the service selection process. However, if the users are also interested in configuring the underlying infrastructure of the cloud service, they need to know more about the resource consumption characteristics of a cloud service.

Current PaaS/IaaS vendors allow customers to configure their PaaS/IaaS offerings based on a predefined set of *resource properties*, e.g. "sizing" the CPU speed and memory capacity of the Virtual Machines or configuring the storage capacity of a database. From the configuration of the resource properties, prices of the PaaS/IaaS offerings can then be calculated accordingly. For instance, Amazon allows their customer to choose among different types of EC2 instances, based on the need of memory, the Elastic Compute Unit (ECU)[6], storage capacity, type of system (32-bit or 64-bit), I/O performance, and the volume of I/O data that can be optimized for persistent storage in the attached Amazon Elastic Block Store (EBS)[7]. Similar to Amazon, Windows Azure provides a way for customers to configure their virtual machines based on the need of CPU speed, RAM, storage capacity and network bandwidth[8]. Private cloud users that use VMWare Workstation to manage their VMs can also specify their VM configurations based on the predefined specification format[9].

Taking into account the need of specifying the resource properties for a PaaS/IaaS, the blueprint approach allows for specifying the *Resource Profile* that can be attached to either a PaaS/IaaS offering or requirement. Definition 3.5 defines a resource profile.

Definition 3.5 (Resource Profile) *A resource profile can be specified for a PaaS/IaaS offering or requirement. A resource profile is a non-ordered collection of resource assertions. Each resource assertion specifies the value range of a resource property.*

By examining the resource specifications of several PaaS/IaaS vendors, the blueprint approach defines the following typical categories of *resource properties*:

- Computing Resource Properties: specify the computing capabilities of a PaaS/IaaS including, for instance, the CPU related properties, memory properties, CPU type, etc.

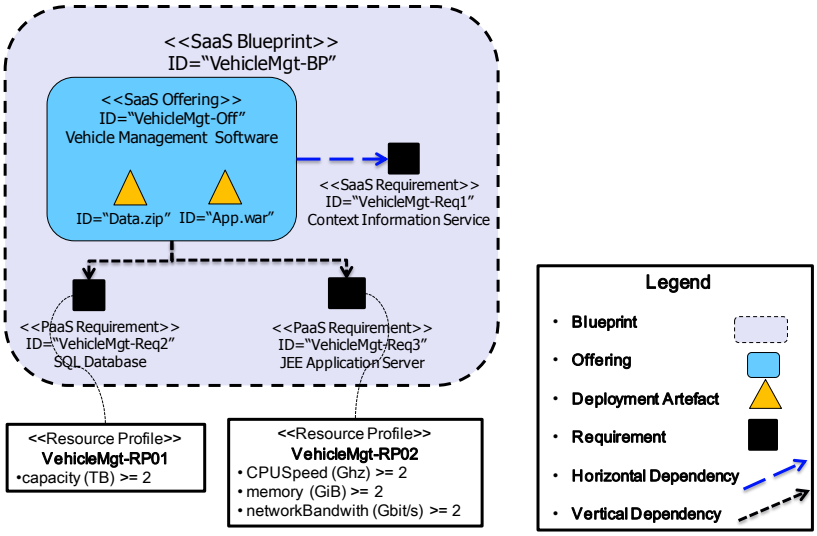
[6]Amazon's Definition of ECU: "We use several benchmarks and tests to manage the consistency and predictability of the performance of an EC2 Compute Unit. One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. This is also the equivalent to an early-2006 1.7 GHz Xeon processor referenced in our original documentation"

[7]Amazon EBS: `http://aws.amazon.com/ebs/`

[8]`https://www.windowsazure.com/en-us/pricing/calculator/?scenario=virtual-machines`

[9]`http://www.vmware.com/support/ws5/doc/intro_vmspec_ws.html#wp1014257`

Figure 3.7: Examples of Resource Profiles in the *Taxi Tilburg Scenario*



- Storage Resource Properties: specify the storage capability of a PaaS/IaaS, including, for instance, the disk storage size, I/O performance, type of disks, etc.
- Network Resource Properties: specify the networking capability of a PaaS/IaaS, including, for instance, the network latency, bandwidth, etc.

With the definition of the categories of resource properties, resource assertions can be specified to describe the value range of a resource property (with a maximum value, or a minimum value, or both), e.g. "$3 \leq CPUSpeed(Ghz) \leq 4$" or "$networkLatency(s) \leq 30$".

Examples of Resource Profiles in the *Taxi Tilburg Scenario*

Examples of resource profiles of the sample `VehicleMgt-BP` blueprint are given in Figure 3.7. The resource profile `VehicleMgt-RP01` specifies the assertion on the storage capacity of the `VehicleMgt-Req2` PaaS requirement. The resource profile `VehicleMgt-RP02` specifies the assertions on the CPU speed, memory size, and network bandwidth of the `VehicleMgt-Req3` PaaS requirement.

3.2.5 Blueprint Classification

A blueprint might be classified according to the following dimensions:

- Classification based on the three layers of the cloud stack: Blueprints can be classified based on the positioning of its cloud service offering on one of the three

layers of the cloud stack: SaaS, PaaS, or IaaS. In this thesis, we often distinguish between the three types of blueprints: a SaaS blueprint containing a SaaS offering, a PaaS blueprint containing a PaaS offering, and an IaaS blueprint contains an IaaS offering.

- Classification based on its Configuration Status - *resolved blueprint* or *unresolved blueprint*: A blueprint has the status "unresolved" if at least one of its requirements is still unmatched, i.e. there exists no horizontal link for this requirement. For instance, the `VehicleMgt-BP` blueprint in Figure 3.5(a) is "unresolved" as it still contains three unmatched requirements.

  In contrast, a blueprint has the status "resolved" if all of its requirements have already been matched, represented by the horizontal links, by at least an offering. Figure 3.5(b) illustrates the "resolved" version of the `VehicleMgt-BP` blueprint showing that all of its requirements already have a horizontal link.

- Classification based on the Marketplace Status - *source blueprint* or *target blueprint*: If a blueprint has already been submitted to a marketplace to indicate that the cloud service is ready to be purchased and reused by the other cloud service providers or CSBA engineers, it is called a *Source Blueprint*. Source blueprints are typically stored in a blueprint repository of a marketplace and publicly accessible for any marketplace users. In contrast, a blueprint specifying an underdeveloping cloud service or CSBA is called a *Target Blueprint*. For instance in the *Taxi Tilburg Scenario*, the `TaxiOrdering-CSBA` is a target blueprint whilst the other blueprints are source blueprints.

  A target blueprint is usually unresolved, and thus requires source blueprints from a blueprint repository to fulfill its requirements. Source blueprints can be both resolved and unresolved. In case an unresolved source blueprint is retrieved from a repository to resolve a target blueprint, the source blueprint iteratively needs to be resolved too.

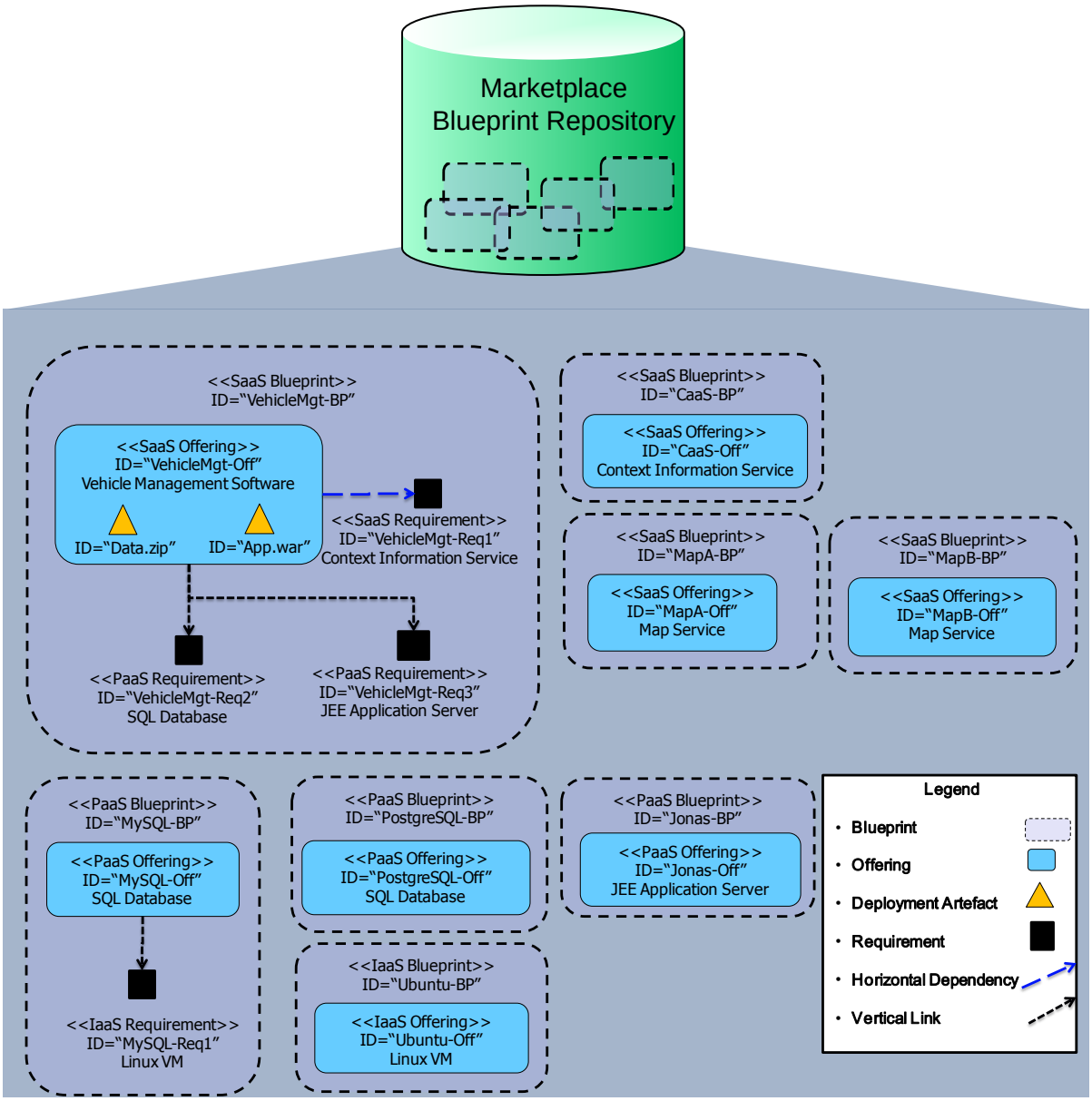
3.3 Blueprint Specification Language (BSL)

The BSL has been developed to support the specification of blueprints as the abstract, uniform cloud service specifications across all layers of the cloud stack. We introduce in the following the main features of the BSL:

- The BSL allows cloud service providers to specify blueprints as the cloud service specifications on all three layers of the cloud stack, i.e. SaaS blueprints, PaaS blueprints, and IaaS blueprints. An CSBA engineer can also use the BSL to specify his CSBA as a composite SaaS blueprint.

- The BSL has been developed based on the blueprint structure defined in the previous section 3.2. Please note that this blueprint structure has been consolidated from a number of desired features proposed by our industry partners.
- The BSL covers both the technical and business aspects of a cloud service specification. The technical aspect includes, for instance, the specifications of the technical interface of a SaaS, the product and technology used to implement a PaaS/IaaS, the deployment artefacts of a cloud service, the required resource capacity for deploying a cloud service, etc. The business aspect is specified in terms of the policy specification for a cloud service, which may include the QoS, cost, licensing, etc.
- The BSL aims to support the design and configuration of an CSBA within the CSBA engineering lifecycle. An CSBA engineer can use the BSL to specify his CSBA design in a SaaS blueprint. Then, his SaaS blueprint can be composed with the other blueprints to configure his CSBA. Finally, the composition of blueprints serves as a manifest to configure the deployment environment of his CSBA.
- The BSL is designed in a modular way to increase the usuability. It contains the following inter-related BSL modules:
  - BSL Core Module: allows to specify blueprints on all three layers of the cloud stack. This module is designed based on the blueprint structure definition in Section 3.2.
  - BSL SaaS Module: extends the BSL Core Module to specify a SaaS blueprint.
  - BSL PaaS Module: extends the BSL Core Module to specify a PaaS blueprint.
  - BSL IaaS Module: extends the BSL Core Module to specify an IaaS bluerpint.
  - BSL Policy Description Module: allows to specify a policy profile that can be associated with an offering or requirement in a blueprint. This module is designed based on the definition of a policy profile in Section 3.2.3.
  - BSL Resource Description Module: allows to specify a resource profile that can be associated with a PaaS/IaaS offering or requirement in a PaaS/IaaS blueprint. This module is designed based on the definition of a resource profile in Section 3.2.4.
  - BSL Interface Description Module: allows to specify the interface descriptions that can be associated with a SaaS offering or requirement in a SaaS blueprint. This module is designed based on existing languages or specification schemas that support the specification of (Web) service interface.

Figure 3.8: Examples of Blueprints in the *Taxi Tilburg Scenario* specified by the BSL



- The BSL is designed in an extensible way so that users can incorporate external languages or specification schemas.

Following the OMG's model-driven engineering methodology [Kleppe et al., 2003], the BSL has been developed as a domain-specific modeling language (DSML) with aim to provide language constructs for specifying blueprints using models. In model-driven engineering, a DSML is a modeling language that is tailored to particular constraints and assumptions of an application domain [Chen et al., 2005]. Following the definition of a DSML in [Chen et al., 2005], the BSL has been developed with the following components:

- The BSL Abstract Syntax: introduces the language constructs to model a blueprint. It is described using the UML class diagram.
- The BSL Concrete Syntax: introduces a concrete XSD template to specify a blueprint. The mapping between the BSL Abstract Syntax and the Concrete Syntax is also provided.
- The BSL Semantic Domain: introduces the formalization of the BSL Abstract Syntax as a knowledge representation schema described by the Web Ontology Language (OWL)- a well-established formal framework for knowledge representation. The mapping between the BSL Abstract Syntax and the Semantic Domain is also provided.

In the subsequent chapter 4, these BSL components will be explained in details.

Examples of Blueprints specified by the BSL in the *Taxi Tilburg Scenario*

Figure 3.8 illustrates some examples of blueprints that have been specified in the *Taxi Tilburg Scenario*. By using the *BSL*, cloud service providers have created a number of blueprints to specify their cloud services including: the `VehicleMgt-SaaS`, the `MapA-SaaS`, the `MapB-SaaS`, the `CaaS-SaaS`, the `JBoss-PaaS`, the `MySQL-PaaS`, the `PostgreSQL-PaaS`, and the `Ubuntu-IaaS`. These blueprints have also been published as source blueprints to the marketplace blueprint repository.

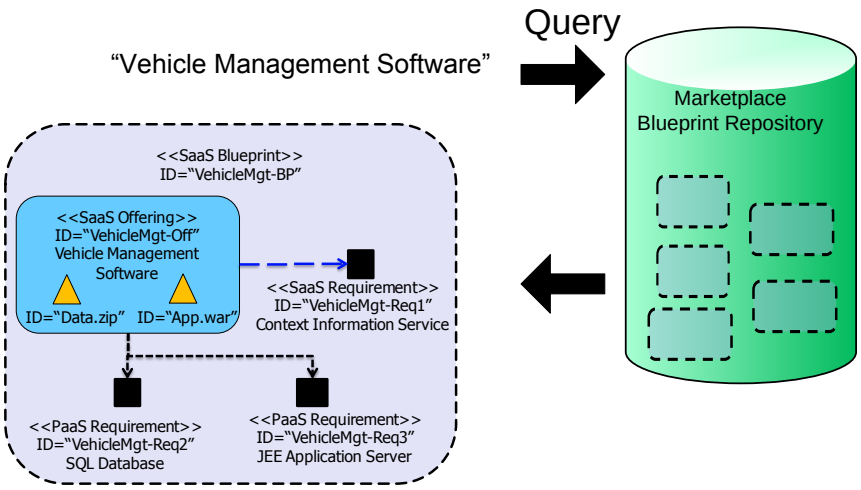
In the next chapter 4, we will provide more details about representing these blueprints using the BSL abstract and concrete syntax.

3.4 Blueprint Manipulation Techniques (BMTs)

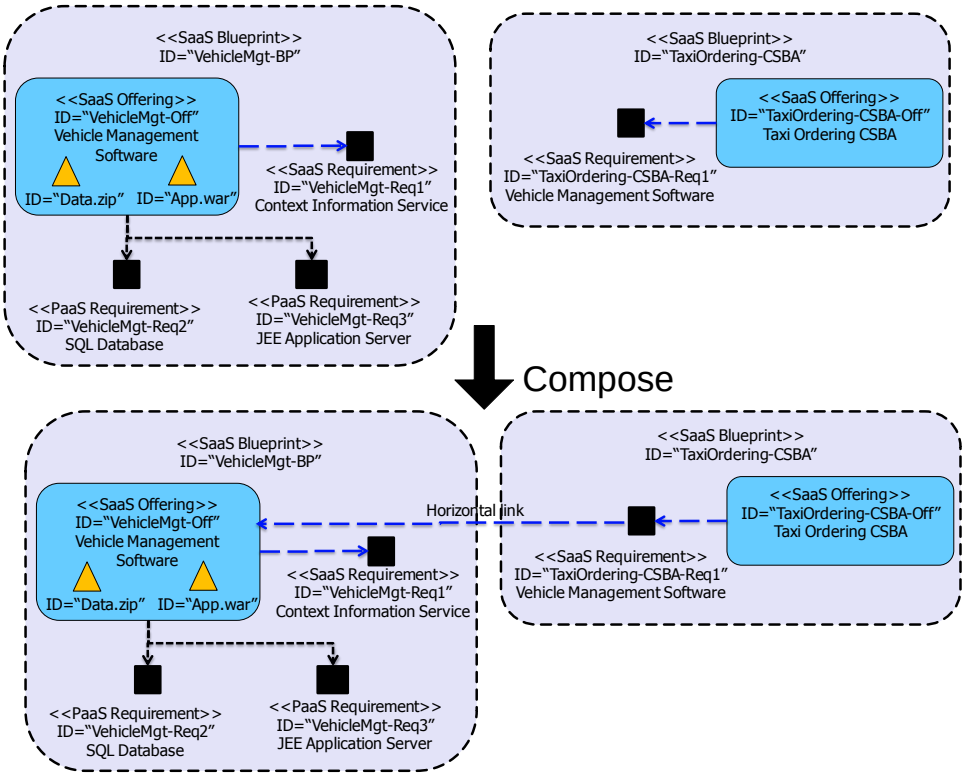
The Blueprint Manipulation Techniques (BMTs) have been developed based on the idea of supporting the generic model operators for manipulating data models in [Melnik, 2004]. In fact, we have taken into account the applicability of these generic operators for the blueprints, selected a subset of them, and extended their definitions towards the requirements of the manipulation techniques that should be used to support the CSBA engineering lifecycle. In particular, the BMTs comprise of the following manipulation techniques:

- Publishing a blueprint to a repository, e.g. a marketplace repository.
- deleting an existing blueprint in the repository.
- Querying blueprints from a repository.

Figure 3.9: Examples of using the BMT in the *Taxi Tilburg Scenario*



Example of Use Case 1: Querying for a blueprint



Example of Use Case 2: Composing 2 blueprints

- Composing blueprints to form a complete CSBA configuration.

In the subsequent chapter 5, a set of operators will be introduced that work directly on the blueprints to support these techniques.

Examples of using the BMT in the *Taxi Tilburg Scenario*

To exemplify the use of the BMT, let us consider the following two sample use cases of the BMT within the *Taxi Tilburg Scenario*, as illustrated in Figure 3.9:

- Case 1- Querying for a blueprint: `TaxiTilburg` is looking for a blueprint from a repository that offers a "Vehicle Management Software" as a SaaS. He submits his request for a "Vehicle Management Software" to the marketplace repository. There is only one blueprint that offers the "Vehicle Management Software" as a SaaS offering, namely the `VehicleMgt-BP` blueprint.
- Case 2- Composing blueprints: After successfully querying the `VehicleMgt-BP` blueprint, `TaxiTilburg` now needs to compose it with his `TaxiOrdering-CSBA-BP` blueprint. Composing blueprints comprises of two steps: (1) identify the matching between a requirement of the `TaxiOrdering-CSBA-BP` blueprint with the offering of the `VehicleMgt-BP` blueprint, and (2) create a horizontal link from the requirement to the offering . Matching between a requirement and an offering does not only include the functional matching, but also include the matching of their policy profiles and/or resource profiles.

3.5 Blueprint Approach in Support of the CSBA Engineering Lifecycle

The aim of this section is to explain how the Blueprint Approach can be used within the CSBA engineering lifecycle to support the design and configuration of an CSBA. Figure 3.10 illustrates the CSBA engineering lifecycle with the support of the Blueprint Approach. Blueprints in this lifecycle are distinguished between source and target blueprints[10], i.e. cloud service providers use *source blueprints* to specify their cloud services and CSBA engineers use *target blueprints* to specify their CSBAs. In Figure 3.10, we also denote where the two components of the blueprint approach, i.e. the BSL and BMTs, are used to support the lifecycle steps.

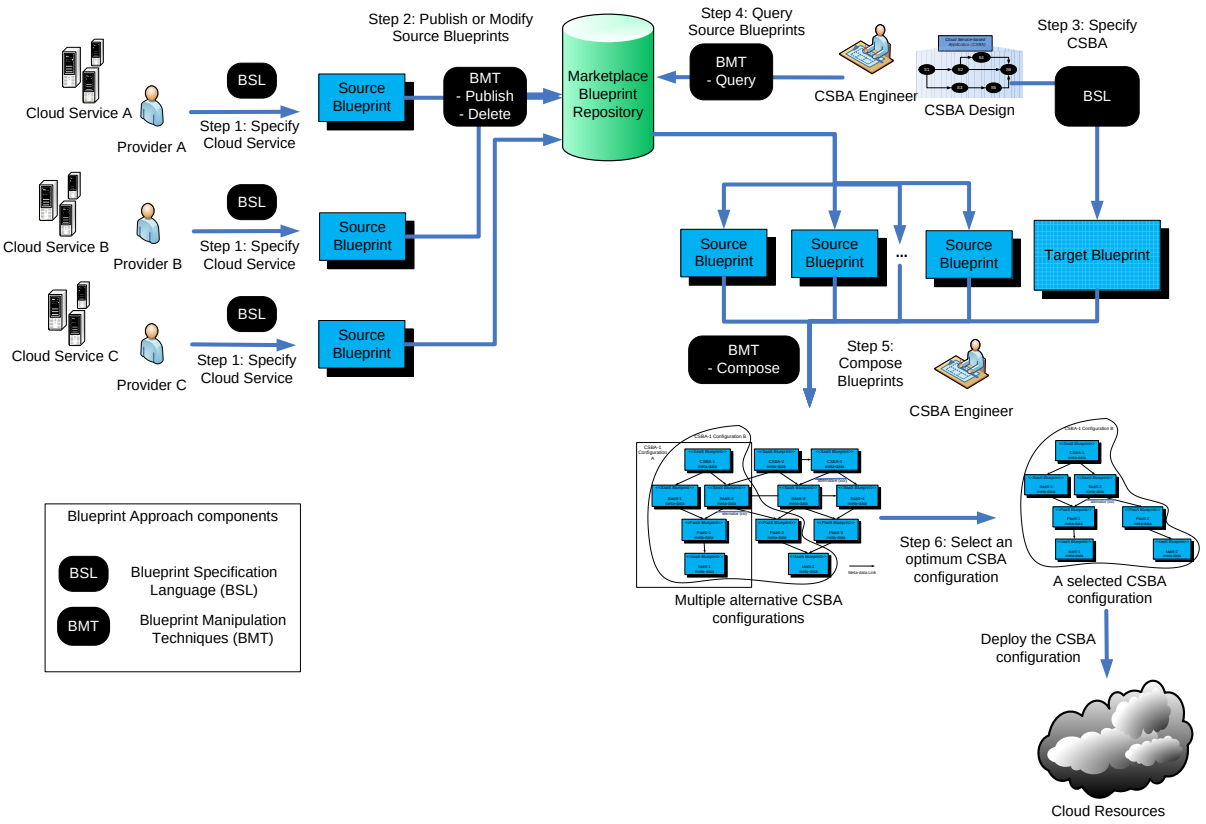
In **Step 1**, cloud service providers would like to commoditize their cloud services. To do so, their cloud services need to be specified in a uniform format and then published in a marketplace repository so that the marketplace users can discover and make use of the offered cloud services. Cloud service providers can use the *BSL* to specify their cloud service in a source blueprint.

In **Step 2**, cloud service providers use the support of the *BMT* to *publish* their source blueprints to a marketplace's blueprint repository. They can also use the *BMT* to *delete* their existing blueprints in the repository.

In **Step 3**, an CSBA engineer would like to develop a new CSBA using cloud ser-

[10]please refer the classification of blueprints based on their marketplace status in Section 3.2.5

Figure 3.10: Blueprint Language Support for CSBA Engineering Life-cycle

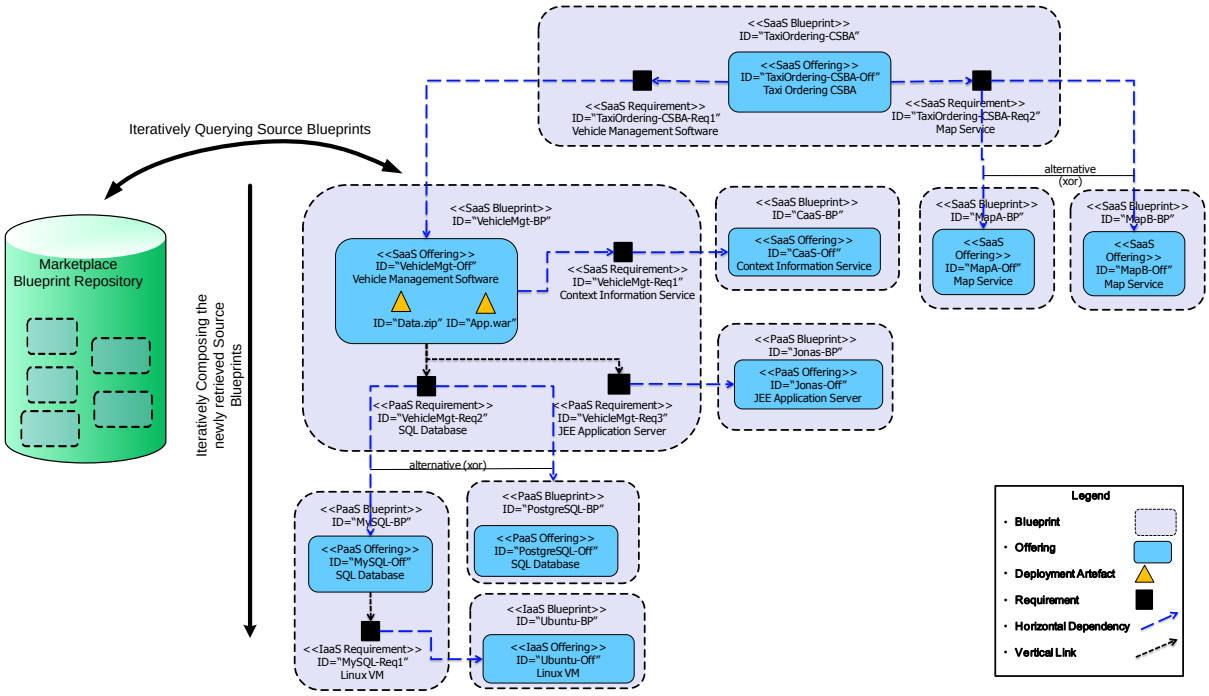


vices. To do this, he uses the *BSL* to specify all the necessary information of his new CSBA. The *BSL* allows the CSBA engineer to describe not only the functionality and policies (e.g., minimum and maximum values for the end-to-end response time and/or availability percentage) of the new CSBA , but also the cloud services required by his CSBA. At the end of this step, the CSBA engineer has completed his CSBA specification in a *Target Blueprint* that contains the CSBA offering and the requirements for third-party cloud services.

In order to transform the Target Blueprint into a complete end-to-end CSBA configuration that can be used to deploy the CSBA on the physical cloud resources, the requirements contained in the target blueprint must be matched by the offerings of the other source blueprints. In Step 1, the Source Blueprints have been specified using the same *BSL* language and in Step 2 they have been published to well-known location(s) like the marketplace's blueprint repository. Therefore, source blueprints can be queried for matching the requirements in the target blueprint. The technique to *query* source blueprints is performed in **Step 4** with the support of the *BMTs*.

In **Step 5**, the target blueprint and a number of source blueprints can *composed* using the support of the *BMTs*. This step results in several alternative configurations of the CSBA, each of which is an end-to-end composition of blueprints.

Figure 3.11: Specifying and Configuring the target blueprint `TaxiOrdering-CSBA-BP`



The next **Step 6** allows the developer to select, among several alternative CSBA configurations, the optimum configuration. Selection criteria might include for instance the cost, the end-to-end performance, the existing resource constraints, or other long-term business strategies. In this thesis, we do not discuss this step of selecting the optimum CSBA configuration. As shown in the lifecycle in Figure 3.10, this step is not supported by the blueprint approach.

Example of following the CSBA engineering lifecycle in the *Taxi Tilburg Scenario*

The company `TaxiTilburg` decides to follow the CSBA engineering lifecycle to configure their TaxiOrdering-CSBA. Figure 3.11 illustrates the result of performing from Step 3 to Step 6 of the lifecycle, namely the specification of the CSBA in a target blueprint called `TaxiOrdering-CSBA-BP` using the *BSL*, and the configuration of the `TaxiOrdering-CSBA-BP` blueprint using the *BMT*.

After creating a target blueprint `TaxiOrdering-CSBA-BP` using the *BSL*, `TaxiTilburg` subsequently uses the *BMT* to *query* for appropriate source blueprints from the marketplace repository whose offerings can be used to match with the requirements of the target blueprint.

Then, `TaxiTilburg` uses the *BMT* to *compose* the target blueprint `TaxiOrdering-CSBA-BP` with the newly retrieved source blueprints, i.e. the `VehicleMgt-BP`, `MapA-BP`, and `MapB-BP` blueprints. The newly retrieved source blueprints may also contain requirements, e.g. the `VehicleMgt-BP` blueprint. Hence,

this querying and composing tasks have to be executed iteratively until the entire configuration of the target blueprint no longer contains any further requirement.

The *Composing* technique supported by the *BMT* is used to compose two blueprints. It comprises of two steps:

- Identify a matching between a requirement of the first blueprint and the offering of the second blueprint. The matching technique does not only take into account the functionality matching, but also the policy and resource matching, i.e. whether the policy assertions specified in the *Policy Profile* attached to the requirement can be satisfied by the policy assertions specified in the *Policy Profile* attached to the offering. For instance, the two SaaS offerings `MapA-Off` and `MapB-Off` can be matched with the SaaS requirement `TaxiOrdering-CSBA-Req2`, because (1) both `MapA-Off` and `MapB-Off` offer the "Map Service" functionality and the `TaxiOrdering-CSBA-Req2` requires a "Map Service" functionality; and (2) the policy assertions specified for the `MapA-Off` and `MapB-Off` regarding the maximum response time and minimum availability can satisfy the policy assertions specified for the requirement `TaxiOrdering-CSBA-Req2`. Furthermore on the PaaS and IaaS layers, matching is also based on the resource specification specified in the *Resource Profiles* , e.g. the `VehicleMgt-BP` blueprint has some resource assertions specified for its PaaS requirement `VehicleMgt-Req2` regarding the required CPU speed, memory size and network bandwidth, and these assertions can be satisfied by the resource assertions of the PaaS offering `JBoss-Off`.

- If a matching can be found between a requirement and an offering, they can be linked by a horizontal link.

It is worth noting in Figure 3.11 that there could be alternative matching results for a requirement, e.g. there are two matching results for the requirement `TaxiOrdering-CSBA-Req2` blueprint and the requirement `VehicleMgt-Req1`. This results in alternative compositions of blueprints to configure the `TaxiOrdering-CSBA`. The next step after configuring an CSBA is to select among the alternative configurations the optimum one to be deployed on the physical cloud infrastructure. The selection strategy might depend on a variety of criteria such as the cost, licensing issues, or some private business constraints. For brevity reason we do not discuss the criteria for selecting an CSBA configuration in this thesis. As an example, a configuration for the `TaxiOrdering-CSBA` that involves the `MapA-BP` blueprint to match the requirement `TaxiOrdering-CSBA-Req2`, and the `MySQL-BP` blueprint to match the requirement `VehicleMgt-Req1` can be selected.

CHAPTER 4

BLUEPRINT SPECIFICATION LANGUAGE

The concept of blueprint has been introduced in Definition 1.4 as a uniform specification for cloud services across all three layers of the cloud stack. Then, the blueprint structure has been defined in Section 3.2. Based on the blueprint structure definition, this chapter introduces a uniform language for specifying blueprints, called the *Blueprint Specification Language (BSL)*.

The chapter is organized as follows:

- Section 4.1 introduces the BSL as a Domain-specific Modelling Language with the following components: the abstract syntax, the concrete syntax, the mapping between concrete and abstract syntax, the semantic domain, and the mapping between the abstract syntax and the semantic domain.
- Section 4.2 introduces the BSL abstract syntax using the Universal Modelling Language (UML) class diagram notations.
- Section 4.3 introduces the BSL concrete syntax as a Blueprint XSD Template. Mapping from the BSL abstract syntax model to the Blueprint XSD Template is also explained in this section.
- Section 4.4 introduces the BSL semantic domain as a Blueprint Schema described by the Web Ontology Language (OWL) - a well-established formal framework for knowledge representation.

4.1 Introduction

Following the OMG's model-driven engineering methodology [Kleppe et al., 2003], the BSL has been developed as a domain-specific modeling language (DSML) with aim to provide language constructs for specifying blueprints using models. In model-driven engineering, a DSML is a modeling language that is tailored to particular constraints and assumptions of an application domain [Chen et al., 2005]. A DSML has been formally defined in [Chen et al., 2005] as a five-tuple of: *Abstract Syntax A*, *Concrete Syntax C*, *Semantic Domain S*, *Semantic Mapping* M_S and *Syntactic Mapping* M_C. The abstract syntax A defines the concepts, relationships, and integrity constraints available in the language. The concrete syntax C defines the specific notation used to express models, which may be graphical, textual or mixed. The syntactic mapping $M_C : C \longrightarrow A$: assigns syntactic constructs (graphical, textual or both) to the elements of the abstract syntax. The semantic domain S is usually defined in some formal framework, in terms of which the meaning of the models is explained. Finally, the semantic mapping $M_S : A \longrightarrow S$: relates syntactic concepts to those of the semantic domain.

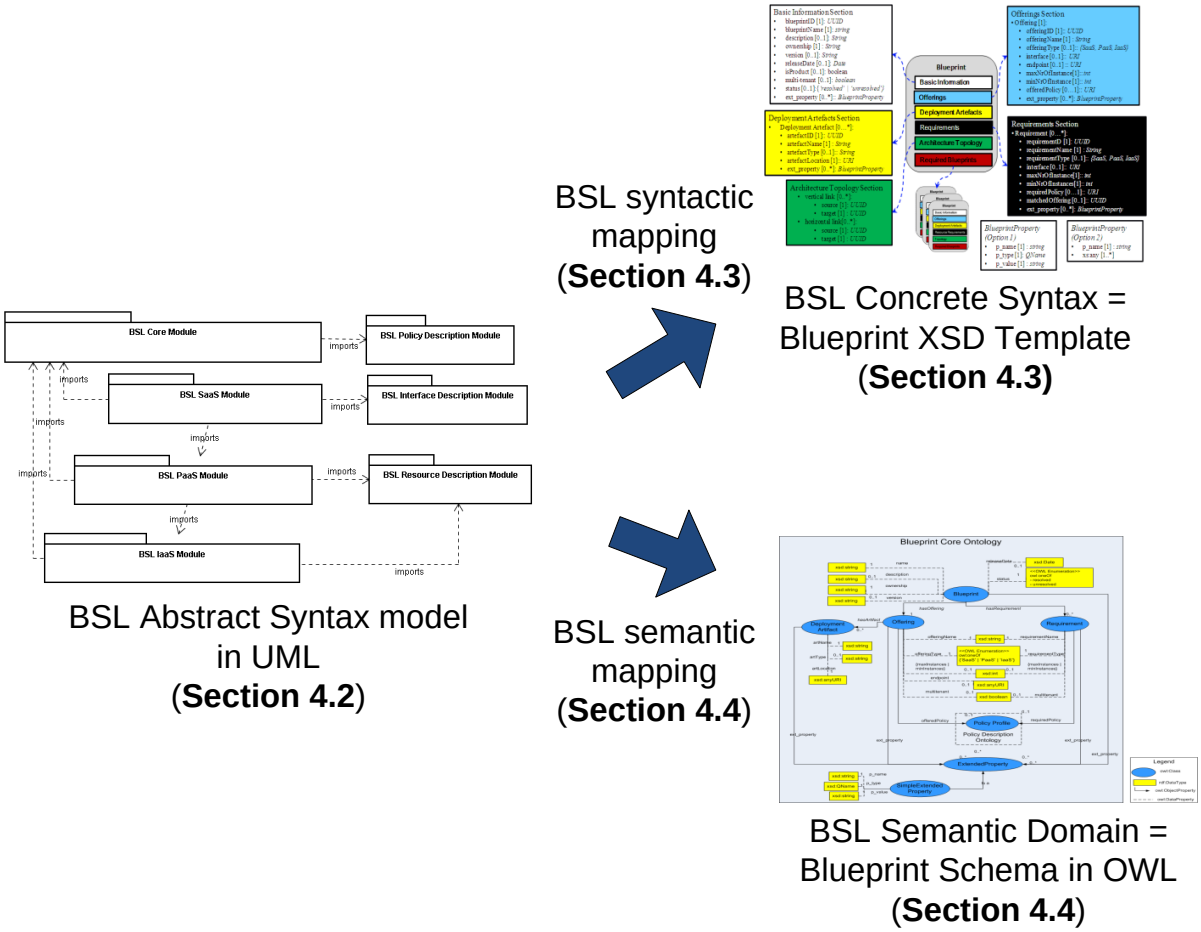
Throughout this chapter, the five components A, C, M_C, S, M_S of the BSL will be subsequently introduced. Figure 4.1 illustrates the content of the chapter showing which BSL components will be introduced in which subsequent section. The languages that are used to specify the BSL components are called meta-languages. In the following, we briefly explain our choices of meta-languages for specifying the BSL components:

- BSL Abstract Syntax: The specification of the BSL abstract syntax requires a meta-language that can express its concepts, relationships, and integrity constraints. Since we are following the OMG's model-driven engineering methodology to define the BSL [Kleppe et al., 2003], we adopt the Universal Modeling Language (UML) class diagram as the meta-language for specifying the BSL abstract syntax. In Section 4.2, we introduce the *BSL model* as the specification of the BSL abstract syntax in UML. The choice of using UML is also aligned with the MOF's four-layer meta-modeling architecture defined by the OMG[1]: the UML is on the M2 layer and the BSL model is on the M1 layer.

- BSL Concrete Syntax and BSL Syntactic Mapping: In the context of cloud computing where blueprints are supposed to be exchanged frequently on the Web, a uniform XML-based representation of blueprints seems to be an appropriate representation format for the data exchange purpose. Hence, we adopt the XML schema (XSD) representation technique to define a concrete syntax for the BSL. Section 4.3 will introduce the *Blueprint XSD Template* as the BSL concrete syntax. The BSL syntactic mapping will also be explained in Section 4.3 as the mapping

[1]Meta Object Facility (MOF): `http://www.omg.org/mof/`

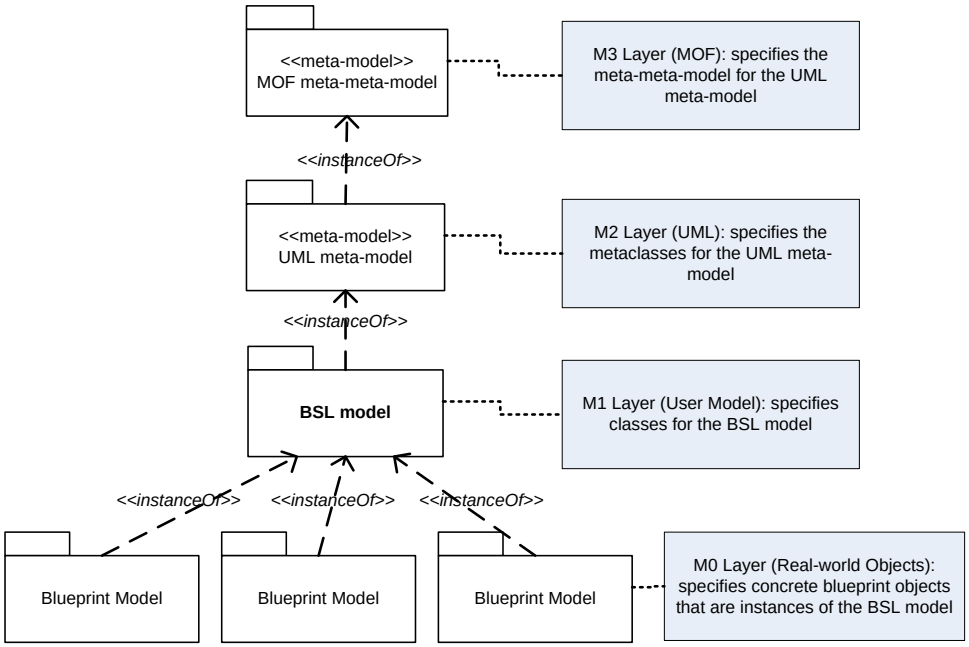
Figure 4.1: Overview of BSL components



from the Blueprint XSD Template to the BSL Meta-model in UML.

- BSL Semantic Domain and BSL Semantic Mapping: According to Emerson et al. [Emerson et al., 2004], the semantic domains and the associated semantic mappings define the *semantics* for a DSML, and these semantics give a precise meaning to those models that we can create using the DSML. Since blueprint itself can be considered as a knowledge-intensive specification of a cloud service [Papazoglou & Vaquero, 2012], an existing formal framework for knowledge representation seems appropriate for the purpose of formalizing the BSL semantics. In Section 4.4, we explain our choice of using the Web Ontology Language (OWL) [McGuinness & van Harmelen (Eds.), 2004] - the existing well-established standard for knowledge representation for Internet-based resources and semantic web- to formalize the BSL semantics. Subsequently, a *Blueprint Schema model described in OWL* will be introduced as the BSL semantic domain. A mapping between the *BSL abstract syntax model in UML* to the *Blueprint Schema Model in OWL* will also be explained in Section 4.4 as the BSL Semantic Mapping.

Figure 4.2: Positioning the BSL Model and the instantiated Blueprint Models within the MOF's meta-modelling architecture



4.2 The BSL Abstract Syntax Model

In this section, the BSL abstract syntax model (*BSL model*, for short) is introduced using the UML class diagram notations to represent the abstract syntax for specifying blueprints. No concrete notations or implementation considerations for the BSL will be mentioned in this section. In Figure 4.2, we position the BSL model on the M1 layer (the User Model layer) of the OMG's MOF meta-modeling architecture. The BSL model is described by the UML (the M2 layer) and aims to provide concepts (described by the UML classes) and relationship between the concepts (described by the UML relations) for modeling blueprints. In fact, there exists another modeling possibility that extends the UML metamodel in the M2 layer with concepts aimed to generate a blueprint model. This solution would be more coherent to standard UML extension mechanism and deliver more UML-consistent models. In the future work, we will reconsider which modeling approach (the M1 or M2 layer) would be the best fit for the BSL model.

Using the BSL model, users can instantiate a *Blueprint Model* that contains concrete blueprint data objects. Alongside with the introduction of the BSL model in this section, we will also provide the examples of using it to instantiate a Blueprint Model containing sample blueprint data objects.

The BSL model is divided in several interrelated *BSL modules* to reduce its complexity and increase the understandability of the language. Figure 4.3 illustrates these BSL modules and the "import" dependencies among them. In the following, we briefly

introduce each BSL module:

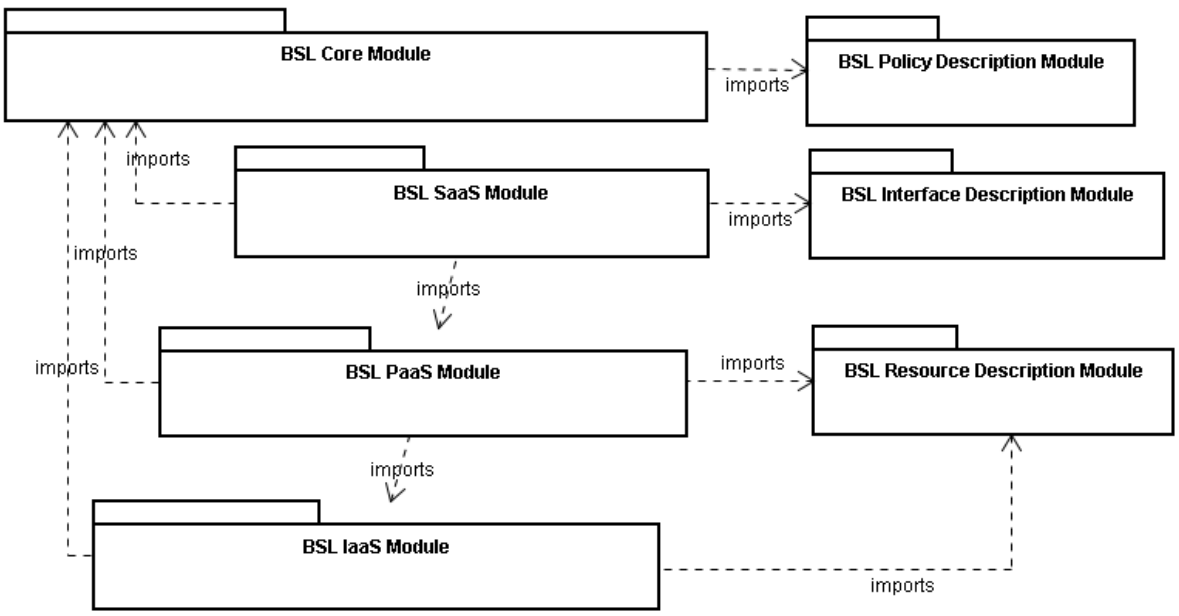
- BSL core module: this module provides the abstract syntax for specifying a blueprint that contains the meta-data elements of a cloud service (i.e. offering, deployment artifacts, requirements, etc.) on any layer of the cloud stack, as well as the links between these elements. Section 4.2.1 will introduce the language model of the BSL core module.

- BSL policy description module: this module provides the abstract syntax for specifying the policy offering/requirement of a cloud service offering/requirement on all three layers of the cloud stack. Hence, this module can be imported into the BSL core module to enable the policy specification in a blueprint. Section 4.2.2 will introduce the language model of the policy description module.

- BSL resource description module: this module provides the abstract syntax for specifying the provided/required resource description of a Paas offering/requirement or IaaS offering/requirement. Hence, this module is imported into the BSL PaaS Module and the BSL IaaS Module to enable the resource specification in a PaaS or IaaS blueprint. Section 4.2.3 will introduce the language model of the BSL resource description module.

- BSL interface description module: This module provides the abstract syntax for specifying the provided/required technical interfaces, i.e. the signature and protocol, of a SaaS offering/requirement. The focus of this module here is to pro-

Figure 4.3: Overview of the BSL Modules



BSL Modules in UML

mote the interoperability between the SaaSs by ensuring the compatibility between the provided technical interface of a SaaS offering and the required interface of a SaaS requirement. Therefore, this module is imported into the BSL SaaS Module to eanble the interface specification in a SaaS blueprint. We reuse the concepts of existing meta-models for specifying (Web) service interfaces to define this language module. Section 4.2.4 will introduce the language model of the BSL interface description module.

- BSL IaaS Module: this module extends the abstract syntax in the BSL core module for the purpose of specifying an IaaS blueprint. Section 4.2.5 will introduce the language model of the BSL IaaS module.

- BSL PaaS Module: this module extends the abstract syntax in the BSL core module for the purpose of specifying a PaaS blueprint. It also imports the BSL IaaS module to allows for specifying an IaaS requirement in a PaaS blueprint. Section 4.2.6 will introduce the language model of the BSL PaaS module.

- BSL SaaS Module: this module extends the abstract syntax in the BSL core module for the purpose of specifying a SaaS blueprint. It also imports the BSL PaaS module to allows for specifying an PaaS requirement in a SaaS blueprint. Section 4.2.7 will introduce the language model of the BSL SaaS module.

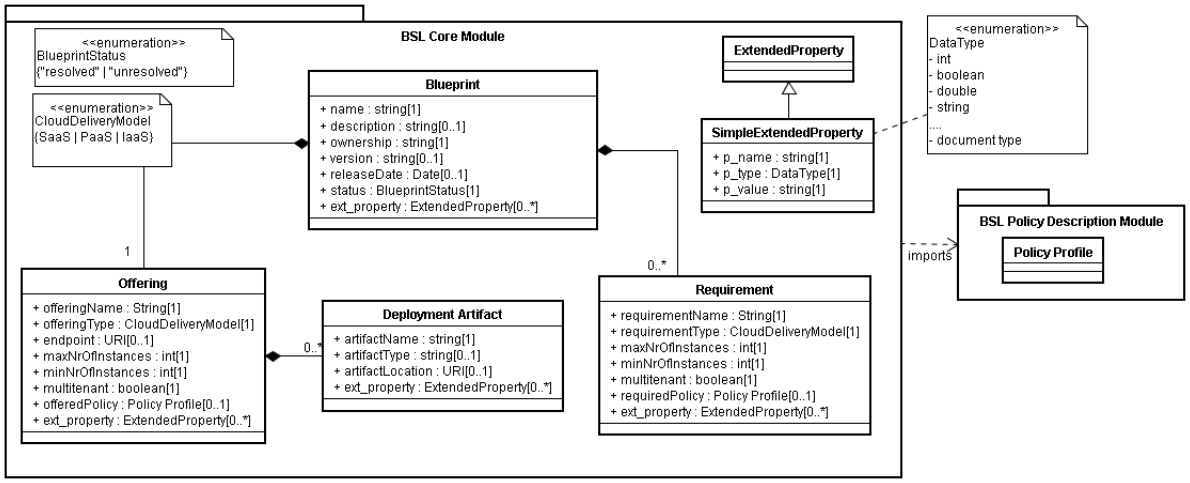
4.2.1 The BSL Core Module

The language model of the BSL core module is presented in Figure 4.4 using the UML class diagram notations[2]. The BSL core module has four essential language concepts, namely *Blueprint, Offering, Deployment Artifact, and Requirement*, corresponding to the concept of the blueprint elements previously introduced in Section 3.2.1. Each language concept in the BSL core module is modeled with a set of predefined *properties*. A property of a concept has either a basic data type, e.g. int, string, boolean, etc., or a complex document types. Within the BSL core module, we use the complex type *Policy Profile* for representing the policy description of a cloud service. The abstract syntax for specifying a *Policy Profile* will be introduced in the BSL policy description module in the next Section 4.2.2. In the following, each concept in the BSL core module and their properties will be explained:

- *Blueprint* is the central concept of the BSL core module as it represents a cloud service meta-data specification provided by a cloud service provider. A blueprint has basic properties including a *name*, an optional short textual *description*, the *ownership* information, the *version* information, the *releaseDate*, and the *status* of

[2]The enumeration class is represented as a comment in this model since the modelling tool we use (named Jude) does not support an explicit construct for modeling an enumeration class

Figure 4.4: The BSL Core Module in UML



the blueprint. The status of a blueprint is defined as its configuration status[3] and is specified by its provider to indicate whether it has been resolved or not. A blueprint consists of blueprint elements, in particular, an *Offering* and a set of zero or many *Requirements*. The composition relationship (dark diamond) between a blueprint and its offering and requirements indicates a part-of integral relationship[4].

- *Offering*: represents the cloud service offered by the blueprint. The offering has the following properties: a mandatory *offeringName* (with type of string) to specify the name of the cloud service offered to the blueprint consumer, an optional *offeringType* to specify the type of the offered cloud service, and an optional *endpoint* (with type of URI) to specify the URI endpoint location for programmatic interactions with the offering. The offered cloud service ranges across three layers of the cloud stack, i.e. it could be a composite CSBA, or a SaaS, PaaS, or IaaS. Inspired by the Debian package management approach [Aoki, 2007], the name of an offering follows a keyword-based approach. This enables an easy searching and matching of offerings that provide the needed cloud services. The type of the cloud service is a simple string specifying the common categorization of cloud services, e.g. it can follow the general classification of cloud abstraction layers "SaaS", "PaaS" and "IaaS", or be more specific like a "database", "application server", "servlet container". The endpoint of an offering might be unknown at the design time of the blueprint. After the blueprint has been fully configured and can be used to deploy the cloud service, this property can be updated to specify the concrete endpoint for the cloud service consumers. The elastic-

[3]Please refer to the classification of blueprints based on the configuration status in the previous Section 3.2.5

[4]This means if the blueprint ceases to exist, all the parts cease to exist too.

ity of an offering is optionally specified in terms of the *minimum and maximum number of instances* of the cloud service that can be offered for each consumer. An optional *Policy Profile* capturing the policy characteristics (including the QoS, regulatory, licensing, security, etc.., characteristics) could be attached to the offering to specify the policy conditions under which the offering is delivered to the consumers.

- *Deployment Artifact*: An offering contains zero or many *Deployment Artifacts*, which are the physical pieces of information that need to be deployed and installed for the provisioning of the offering. The information of a deployment artifact encompasses: a mandatory *artifact name*, an optional *artifact type* indicating whether this artifact is a software binary, composition script, database configuration and startup file, or some other kinds of configuration files. An optional *artifact location* can be specified to indicate the URI location from which the artifact can be downloaded and installed.

- *Requirement*: The abstract syntax for specifying a requirement is similar to the one used for specifying an offering, except that a requirement obviously has no endpoint. In particular, a requirement has the mandatory *requirementName* and *requirementType* properties to indicate the name and type of a particular cloud service needed by a blueprint. A requirement may be optionally specified with the required minimum and maximum number of instances that should be provided, and an optional *Policy Profile* to specify the policy constraints under which the needed cloud service should be provided.

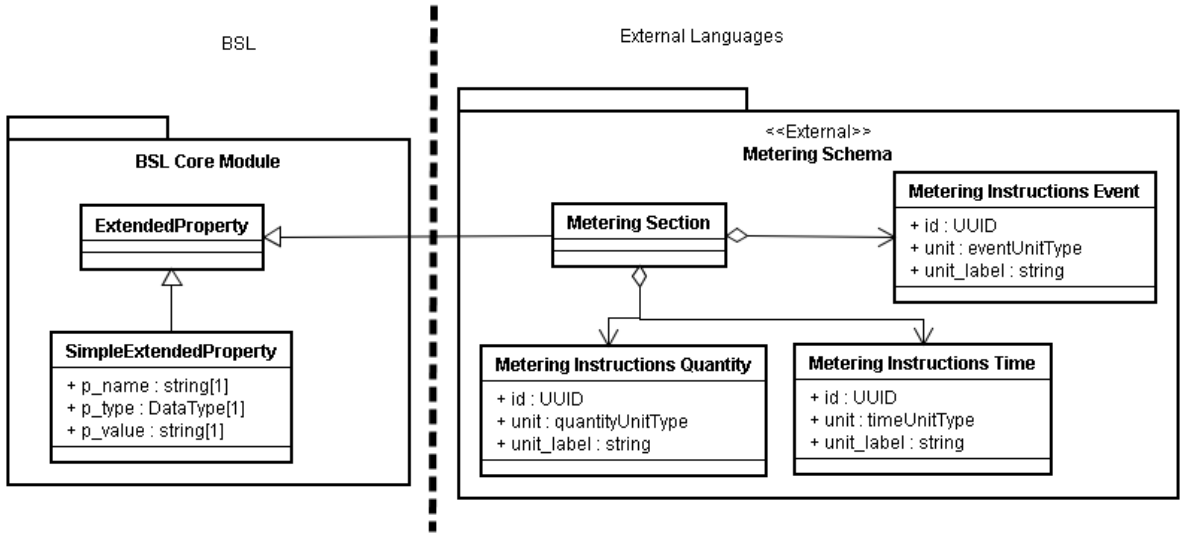
Since the offering and requirement classed in the BSL core module are quite identical, another modeling option is to define an abstract class as a super class capturing their commonalities. Then, the offering and requirement classes only need to specialize their differences. This modelling option would indeed deliver a cleaner model for the BSL and will definitely be considered in the next improvement cycle of the language.

Extensibility of the BSL core module

The BSL core module introduced in this section defines a generic blueprint schema for specifying all categories of cloud services on all cloud abstraction layers "SaaS", "PaaS" and "IaaS". This module will be extended in the subsequent sections 4.2.7,4.2.6, and 4.2.5 to specify the blueprint schema for cloud services on each of these three layers of the cloud stack.

However in general, cloud services may be further classified into different specific categories, i.e. logistics SaaS, database PaaS, machine IaaS, and may thus need to be specified with additional properties for which the blueprint schema defined by the BSL modules may not be sufficient. We acknowledge this important requirement

Figure 4.5: Extending the BSL with the external Metering Schema

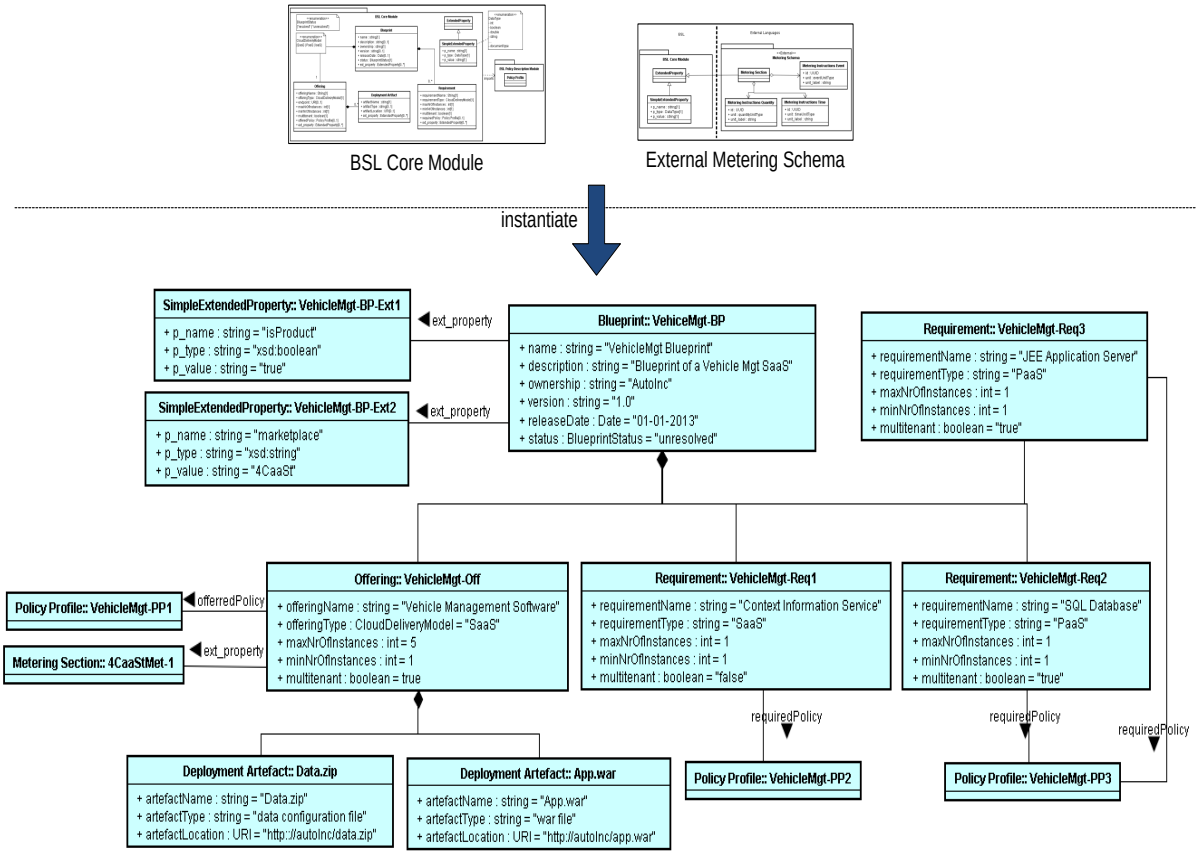


and thus design our BSL core module in an extensible way, i.e. the BSL core module in Figure 4.4 allows for defining category-specific properties *ext_property* of type *ExtendedProperty* for all the language concepts *Blueprint, Offering, Deployment Artifact, or Requirement*. The type *ExtendedProperty* can be considered as the extension point of the BSL, through which a cloud service provider can extend the language constructs following the two ways:

- The BSL core module provides a simple way for extending its language construct by allowing the category-specific properties *ext_property* to be defined with the type definition *SimpleExtendedProperty*. The type *SimpleExtendedProperty* extends the type *ExtendedProperty* and consists of a property name (*p_name*), property type (*p_type*), and property value (*p_value*). As an example, if a blueprint provider wants to specify that his offering is already a product listed on the marketplace "4caast" and can only be purchased via a "4caast" contract, he could extend his offering with the following two simple extended properties: {p_name='isProduct', p_type='xsd:boolean', p_value='true'}, and {p_name='marketplace', p_type='xsd:string', p_value='4caast'}.

- Another way to extend the language constructs of the BSL is to import the external language schemas and specify that the external language constructs extend the *ExtendedProperty* concept. Figure 4.5 presents a sample BSL extension that supports the definition of *Metering Section* as an *ext_property*. This BSL extension has been defined for the 4caaSt marketplace [European Comission, 2010] to allows the blueprint providers to specify the metering information for their blueprints, which is needed for accounting purpose as soon as their blueprints are published and purchased on the 4CaaSt marketplace.

Figure 4.6: Example of a Blueprint specified by the BSL Core Module

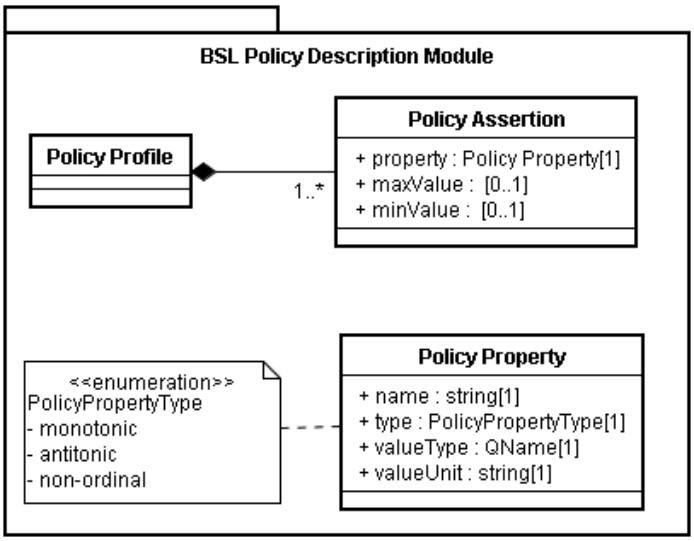


Blueprint Model specifying a Blueprint

Example of a Blueprint specified by the BSL Core Module

By instantiating the language concepts and their relations in the BSL core module, a *Blueprint Model* can be created for the purpose of specifying blueprints. Figure 4.6 depicts a sample Blueprint Model for specifying the sample `VehicleMgt-BP` blueprint that has been instantiated from the BSL Core Module. This model contains instances of the BSL concepts, which have been assigned with a unique ID during the instantiation, e.g. the unique ID of the blueprint is *VehicleMgt-BP*. The details of the `VehicleMgt-BP` blueprint have already been introduced in the previous chapters and thus will not be repeated here. Instead, we would like to explain the two alternative usage of the extended properties supported by the BSL Core Module. In Figure 4.6, the `VehicleMgt-BP` blueprint is specified with two *simple extended properties*: the `VehicleMgt-BP-Ext1` for indicating whether the blueprint is already a product and the `VehicleMgt-BP-Ext2` for indicating the marketplace where the blueprint can be found. Instead of using the *simple extended properties* for extending a BSL language construct, the `VehicleMgt-Off` offering is extended with an external language construct, i.e. the `4CaaStMet-1` is an extended property of type `Metering Section` supported by the external `Metering Schema` that has been introduced previously in Figure 4.5.

Figure 4.7: The BSL Policy Description Module in UML

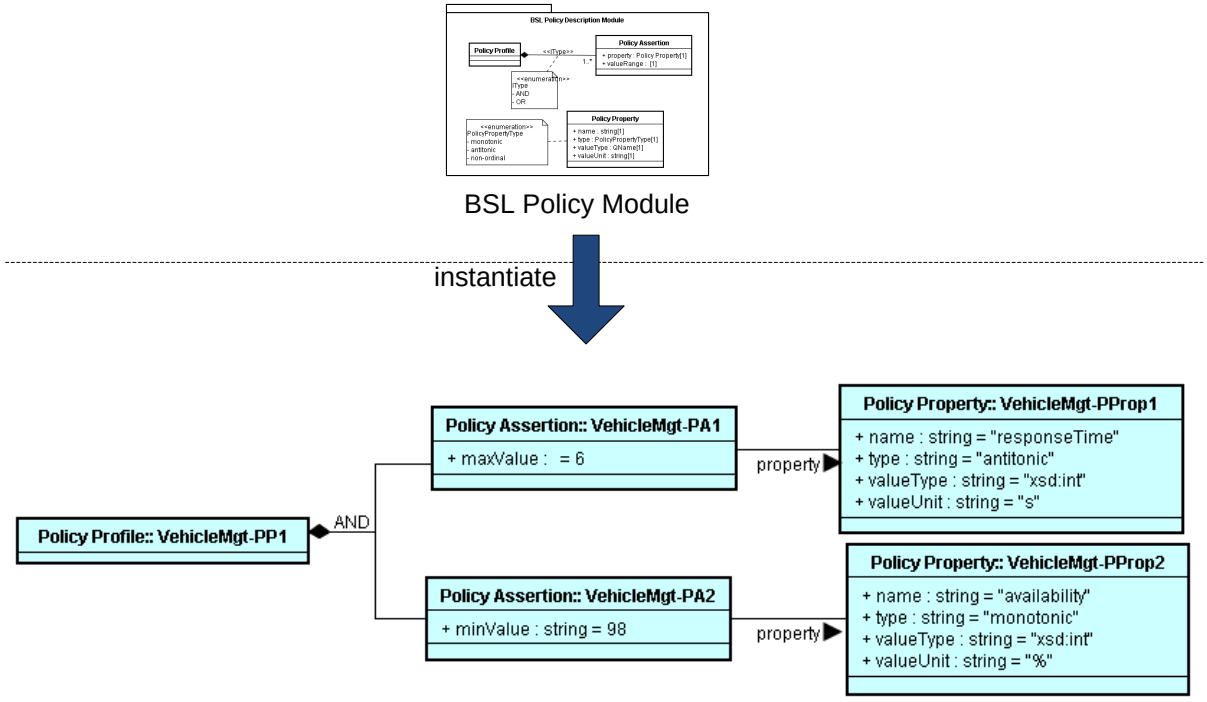


4.2.2 The BSL Policy Description Module

The BSL policy description module illustrated in Figure 4.7 has been developed as a simplified version of the WS-Policy language [W3C, 2006]. It defines the concept of a *Policy Profile* to capture the promised policy capabilities of an offered cloud service or the policy requirements of a required cloud service. Following the Definition 3.4 in the previous Section 3.2.3, policy profile is defined as a non-ordered collection of *policy assertions*. A policy assertion contains a statement about a *Policy Property*, e.g. availability, response time, throughput, security, privacy etc., associated with a *maximum value* and a *minimum value* and , e.g. "minValue = 98%" for the availability, and "maxValue = 6s" for the response time in seconds. A policy property is of type "monotonic", "antitonic", or "non-ordinal". Monotonic policy properties order their values with increasing order, e.g. availability and throughput; antitonic policy properties order their values with decreasing order, e.g. response time, failure rate; and non-ordinal policy properties are those whose values cannot be ordered without further specific ordering criteria, such as security, licensing, privacy, etc. The *maxValue* is used to specify the value of a non-ordinal policy property, e.g. "maxValue= Basic256Rsa15" for the security.

The BSL concept *policy profile* is imported to the BSL core module, to enable the specification of policy capabilities for an offering or policy requirements for a requirement in a blueprint. Since WS-Policy is a well-known language for specifying service policy, the decision to develop the BSL Policy Module as a simplified version of the WS-Policy language has also another benefit regarding the interoperability with existing service description languages.

Figure 4.8: Example of a Policy Profile specified by the BSL Policy Description Module



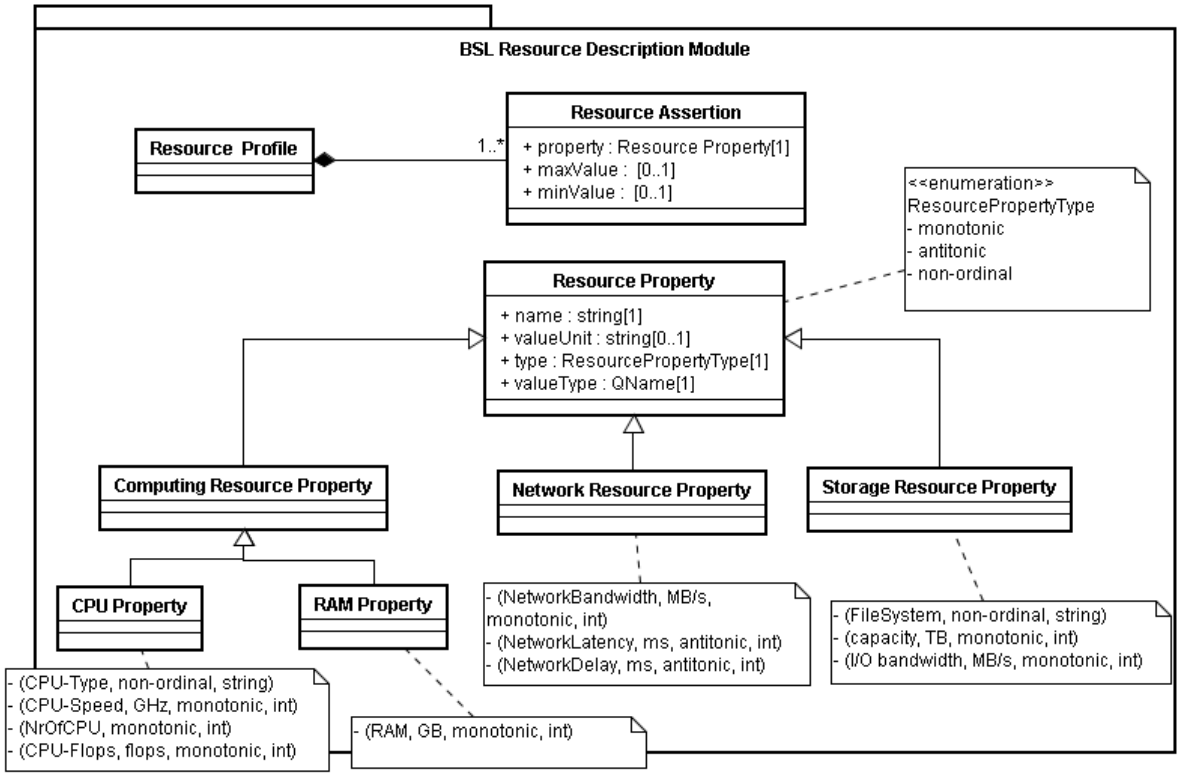
Example of a Policy Profile specified by the BSL Policy Description Module

By instantiating the language concepts and their relations in the BSL policy description module, a Blueprint Model can be created for the purpose of specifying a policy profile. Figure 4.8 depicts a sample Blueprint Model containing the `VehicleMgt-PP1` policy profile that, according to our running example, is attached to the `VehicleMgt-Off` offering to indicate its policy capabilities. Two policy assertions are specified in the `VehicleMgt-PP1` policy profile, one for specifying the maximum response time and the other for specifying the minimum availability of the cloud service offering.

4.2.3 The BSL Resource Description Module

The BSL resource description module introduced in Figure 4.9 defines the concept of a *Resource Profile* to capture the offered resource capabilities of an offered PaaS/IaaS or the required resource capabilities of a required PaaS/IaaS. From a first glance, the definition of the BSL resource description module looks similar the BSL policy description module. It is indeed the case since we have also reused the structure of the WS-Policy language [W3C, 2006] for the development of the BSL resource description module. There is a small extension in this module in the way that we propose also a classification of types of resource properties: Computing, Network, and Storage, which in our

Figure 4.9: The BSL Resource Description Module in UML

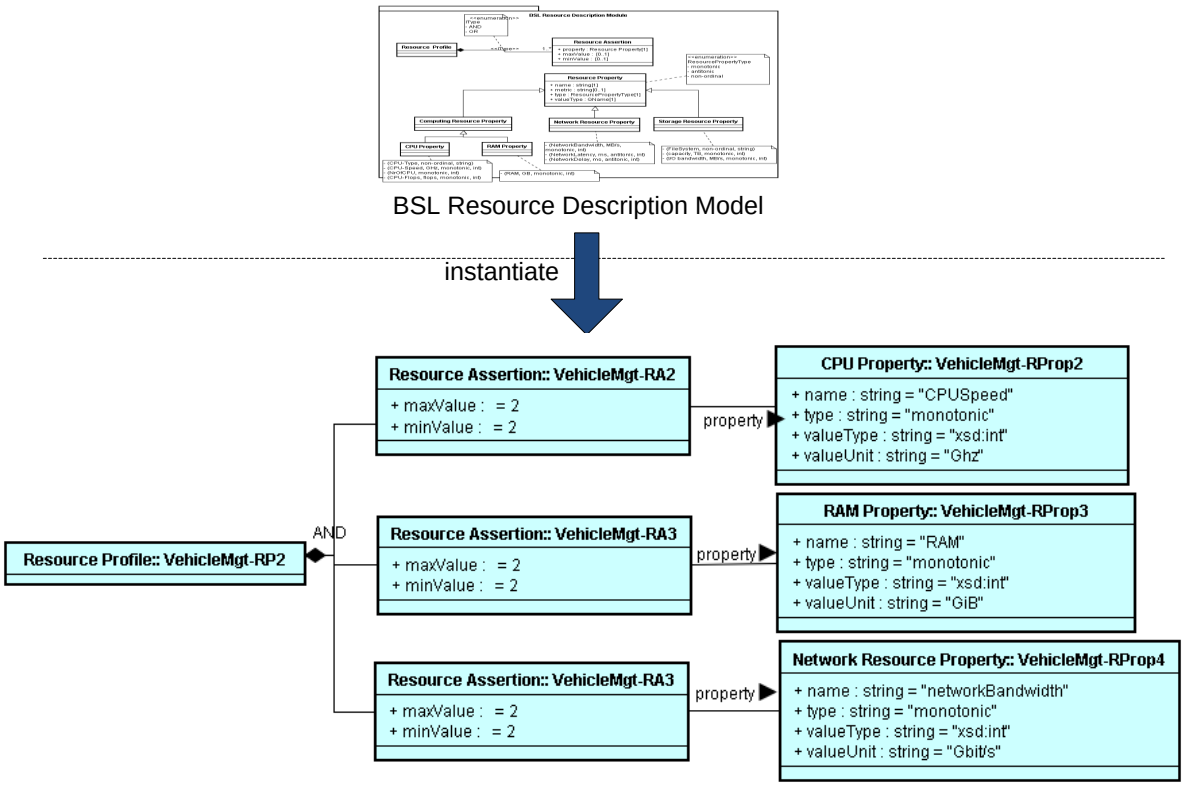


opinion covers the most common resource properties of a cloud service.

A resource profile has been defined in Definition 3.5 in the previous Section 3.2.4 as a non-ordered collection of *resource assertions*. Each resource assertion contains a statement about a *Resource Property*. This BSL module also provides a predefined categorization of resource properties, in which we classify them into *Computing Resource Properties*, e.g. the CPU-type, CPU-Speed, RAM, etc., Storage Resource Properties, e.g. capacity, I/O bandwidth, etc., and *Network Resource Properties*, e.g. NetworkLatency, NetworkBandwidth, etc. An assertion specifies a resource property with a *maximum value* and a *minimum value*, e.g. "minValue = 2" for the CPU-speed in Ghz, "minValue = 2" for the network bandwidth in Gbit/s, etc. The resource property can be of type "monotonic", "antitonic", or "non-ordinal". Monotonic resource properties order their values with increasing order, e.g. network bandwidth and CPU-Speed; antitonic resource properties order their values with decreasing order, e.g. network latency, network delay, etc.; and non-ordinal resource properties are those whose values cannot be ordered without further specific ordering criteria, e.g. the CPU type. The *maxValue* is used to specify the value of a non-ordinal resource property, e.g. "maxValue= Intel Dual Core" for the CPU type.

The BSL concept *resource profile* is imported into the BSL PaaS module and the BSL IaaS module to support the specification of the offered or required resource capabilities

Figure 4.10: Example of a Resource Profile specified by the BSL Resource Description Module



for a PaaS/IaaS offering or a PaaS/IaaS requirement respectively.

Example of a Resource Profile specified by the BSL Resource Description Module

By instantiating the language concepts and their relations in the BSL resource description module, a Blueprint Model can be created for the purpose of specifying a resource profile. Figure 4.10 depicts a sample Blueprint Model containing the `VehicleMgt-RP2` resource profile that, according to our running example, is attached to the `VehicleMgt-Req3` requirement (i.e. the requirement for a JEE application server) to indicate the required resource capacities of this requirement. Three resource assertions are specified in the `VehicleMgt-RP2` resource profile for specifying the required CPU speed, memory size, and network bandwidth.

4.2.4 The BSL Interface Description Module

On the SaaS layer, a SaaS offering is a provided cloud software service and a SaaS requirement is a required cloud software service in a blueprint. The interfaces of

a SaaS need to be described in a consistent manner so that they can be discovered and composed easily across blueprint providers. Existing approaches for describing component interfaces and web service interfaces could be revised and then reused for describing cloud service interfaces. In its early time, component-based system has already realized the need of having a uniform description of component interface, e.g. the IDL in the CORBA framework, that comprises of a set of operations with input, output and exception parameters. A comprehensive description of a component interface has been proposed in [Beugnard et al., 1999] that has four levels: syntactic, behavioral, synchronization and QoS.

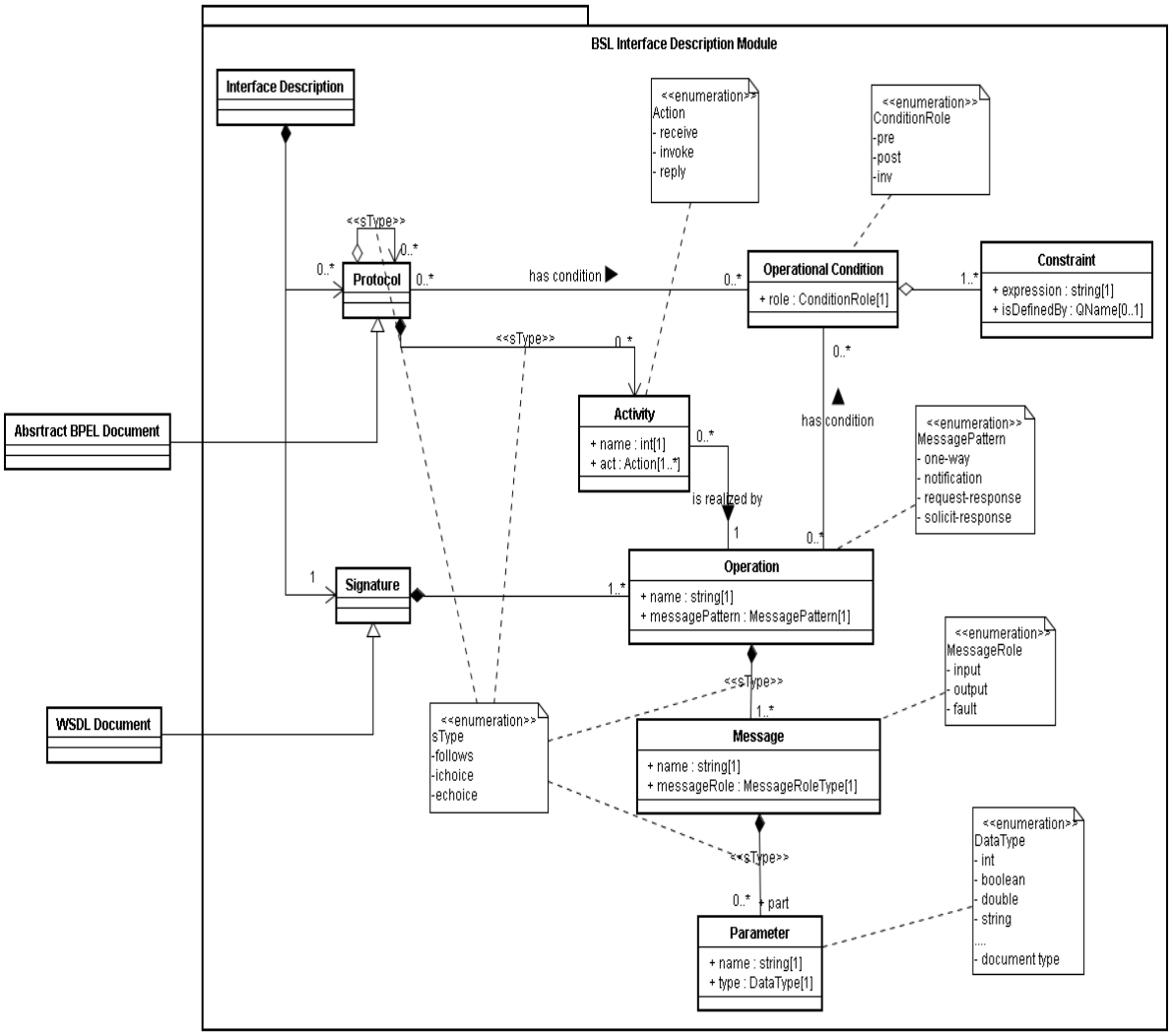
The de-facto standard for describing web service interfaces is the WS-* family, which basically consists of a number of XML-based specification schemas, e.g. WSDL schema for specifying a *Structural Interface*, Abstract BPEL schema for specifying an *Behavior Protocol*, and the WS-Policy schema for specifying a *Policy Profile*. Other initiatives like the CBDi Reference model [Everware-CBDI,] or OASIS SOA reference model [OASIS, 2006] provide a complete technology-agnostic service description, i.e. no specific technologies are required for describing services. Apart from the service interface, these reference models cover also other aspects of services.

Another approach for describing software service interface in a technology-agnostic way has been presented in [Andrikopoulos, 2010] using an underpinning Abstract Service Description (ASD) meta-model as the specification schema. An instantiated ASD is a service interface description containing the following layers, according to its meta-model in [Andrikopoulos, 2010]:

- structural layer: contains the method signatures and their message parameters required for the interaction of the clients with the service
- behavioral layer: contains the records describing the perceived behavior of the service in terms of exchange of messages grouped under service operations, and the conditions under which message exchanges may occur.
- non-functional layer: adopts the simplified version of the WS-Policy schema for the description of QoS characteristics in the forms of assertions that are associated with the services.

Given that the non-functional layer of this ASD meta-model has been captured in our BSL Policy Description Module in Section 4.2.2, we decide to develop the BSL Interface Description module based on its structural and behavioral layers. The reason for reusing the ASD meta-model is because it aggregates the concepts found in widely adopted technologies like WSDL and BPEL, with the ones from higher-level description models like the OASIS SOA reference model [OASIS, 2006]and the CBDI SOA meta model [Everware-CBDI,]. Furthermore, our aim is not to invent yet-another-language for specifying service interfaces, but only to find a solid technology-agnostic

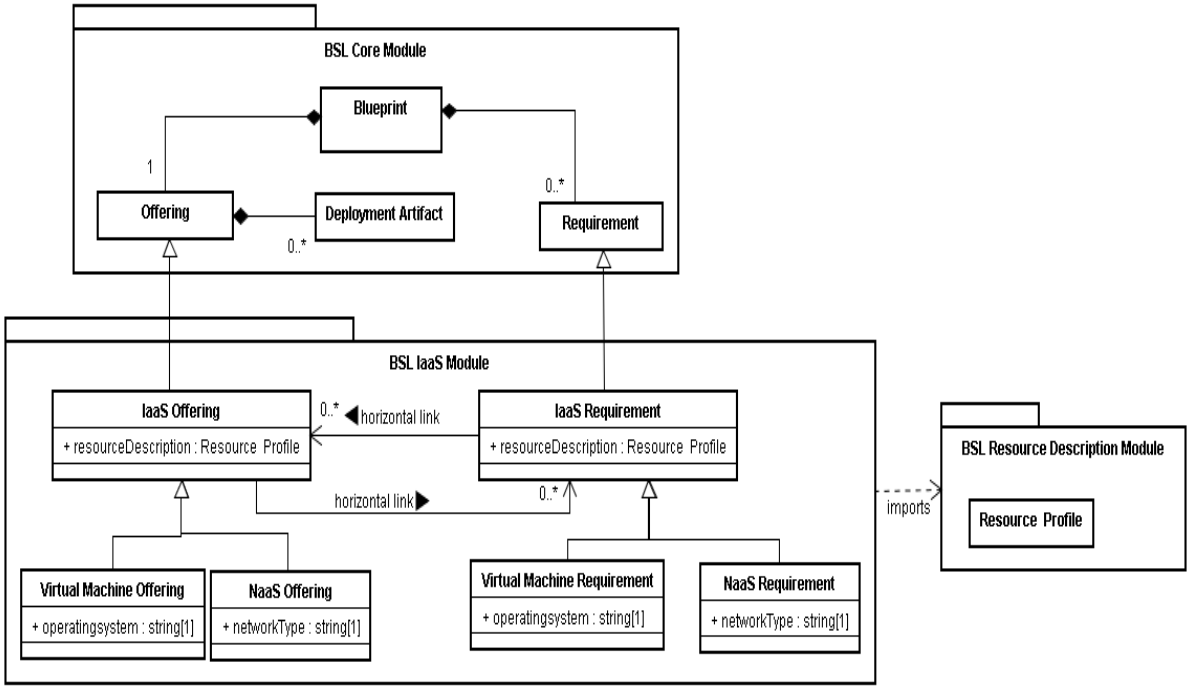
Figure 4.11: The BSL Interface Description Module



representation of cloud service interfaces that serves as a significant module of the BSL model.

Figure 4.11 introduces the BSL interface description module for specifying the interfaces of a SaaS. In this module, an *Interface Description* of a service consists of at least a *Signature* and zero or more *Protocol*. A *signature* of a service defines the static description of a service with its operational semantics. In particular, it is composed of *operations*, i.e., specific functions that a service provides by consuming input messages and producing output or fault messages. An operation has a name and follows one of the following messaging patterns: one-way(i.e. input only), notification (i.e. output only), request-response (i.e. input-output), or solicit-response (i.e. output-input). Input, output, and fault *messages* are logically organized in typed *parameters*. The parameter type could be a simple type, e.g. int, boolean, string etc., or a complex document type.

Figure 4.12: The BSL IaaS Module in UML



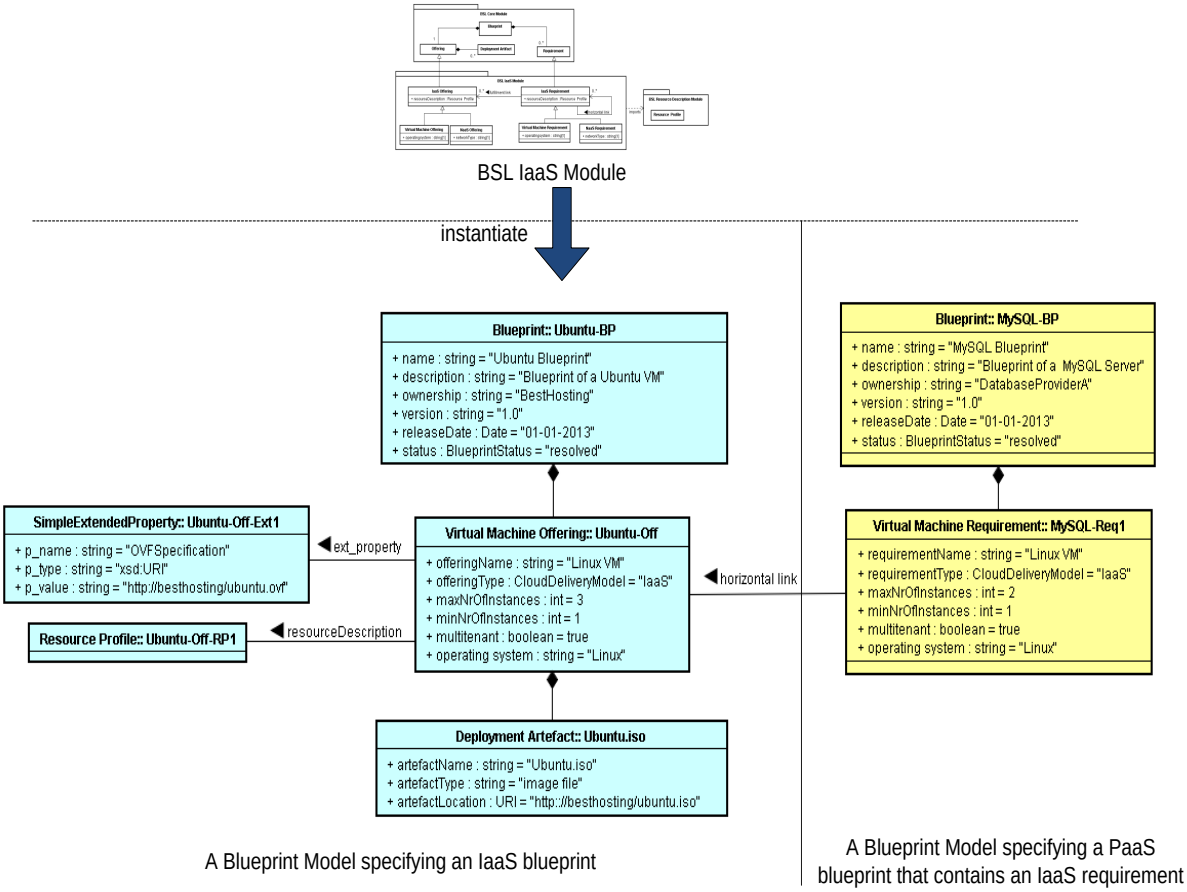
A *protocol* is designed on top of the signature, managing the ordering aspect of executing service operations. In particular, it may encompass a number of *activities* organized according to the sequencing relationship *sType*, i.e. an activity can follow, be concurrent (ichoice) or be exclusive (echoice) with another activity. A protocol can contain nested protocols that are also organized following the sequencing relationship *sType*. An activity entails a logical functionality that has a name and performs an action, which could receive an incoming service request for an operation and then reply with a response, or invoke an external operation of another service. *Operational Conditions* are defined as sets of *constraints* and then attached to an operation or behavioral protocol as the pre- and post-, and invariant-conditions of its execution. A constraint specifies a particular operational condition through a string-based expression and a URI indicator of the constraint language used to formulate the constraint expression.

For brevity reason, we do not introduce an example of using the BSL Interface Description module as we assume that the interface of a SaaS blueprint in the *Taxi Tilburg Scenario*, e.g. the `VehicleMgt-BP` blueprint, can be specified using a WSDL document and a BPEL document.

4.2.5 The BSL IaaS Module

The BSL IaaS module in Figure 4.12 extends the BSL core module by specifying the two types of IaaS offering/requirement, the virtual machine offering/requirement and

Figure 4.13: Example of an IaaS blueprint specified by the BSL IaaS Module



Network-as-a-service (NaaS) offering/requirement. Virtual machine offerings/requirements are specified with information about its operating system and resource capacity. NaaS offerings/requirements are specifies with information about its network type and network resource capacity. The resource capacity is specified in terms of the *Resource Profile*, which is a concept imported from the BSL Resource Description Module presented in the previous Section 4.2.3.

Regarding the dependency links between the elements of an IaaS blueprint, the BSL IaaS module supports the specification of:

- A "horizontal link" between an IaaS offering and an IaaS requirement within the same IaaS blueprint. This horizontal link indicates the need for a cross-IaaS integration.

- A"horizontal link" between an IaaS requirement and an IaaS offering of two distinctive blueprints. This horizontal link indicates that the IaaS offering can be reused to fulfill the IaaS requirement, and thus is usually used to compose an IaaS blueprint with other blueprints.

It is easy to recognize that the BSL IaaS module introduced in this section is very simple. Given a number of existing approaches that already focus on the specification format for an IaaS, e.g. the OVF [DMTF, b] or OCCI [OCCI-Working Group, 2011], our blueprint definition for an IaaS is very minimum. Nevertheless, by taking into account the extensibility of the BSL core module, existing specification standards like the OVF [DMTF, b] for specifying the packaging and distribution of virtual appliances and OCCI [OCCI-Working Group, 2011] for specifying the management APIs, can easily be incorporated into our blueprint definition for an IaaS through the extended properties. As an example, Figure 4.13 will introduce an IaaS offering that is specified with a reference to its packaging information stored in an OVF file.

Example of an IaaS blueprint specified by the BSL IaaS Module

By instantiating the language concepts and their relations in the BSL IaaS module, a Blueprint Model can be created for the purpose of specifying an IaaS blueprint. Figure 4.13 depicts a sample Blueprint Model for specifying the `Ubuntu-BP` blueprint that, according to our running example, is an IaaS blueprint containing the `Ubuntu-Off` virtual machine offering. Linux operating system is used in the virtual machine offering and the resource capacity of the virtual machine is specified in the `Ubuntu-Off-RP1` resource profile. The ISO image file of the virtual machine is specified as a deployment artefact in the blueprint. To enable portability across different infrastructure cloud, the packaging information of this IaaS offering is specified in a separate OVF file that is referenced through a URI by the *simple extended property* `Ubuntu-Off-Ext1`.

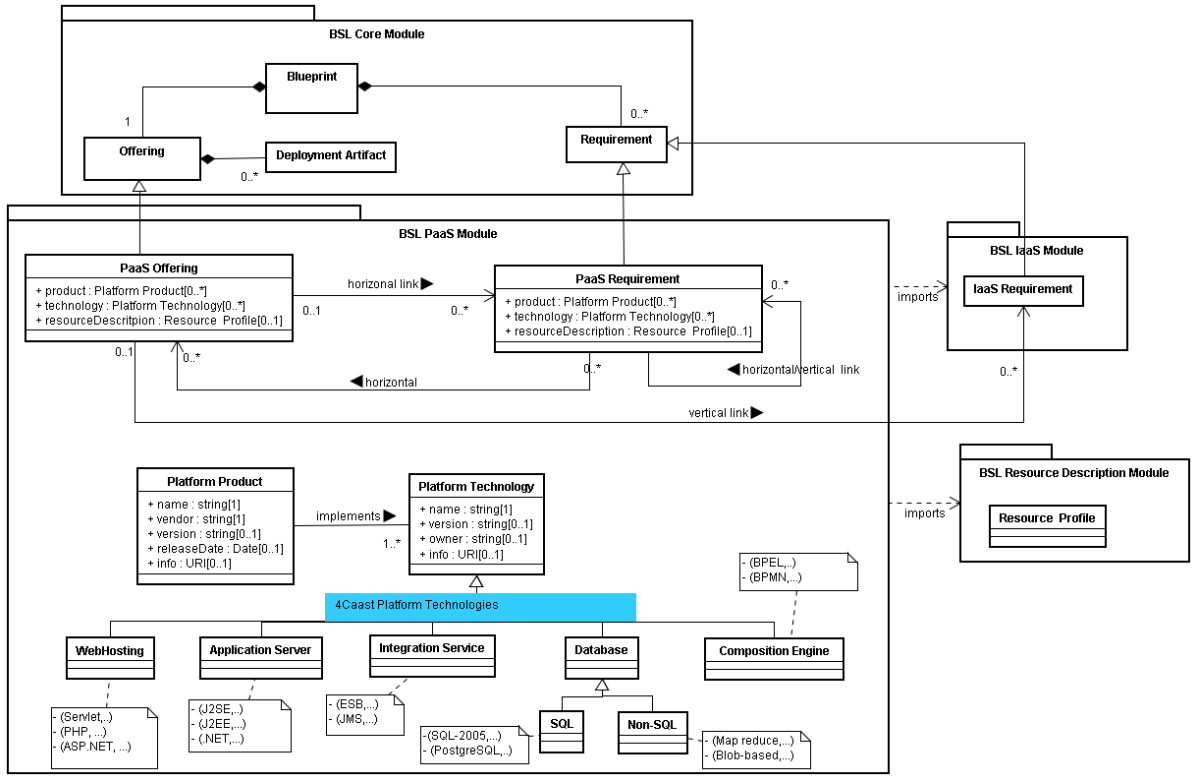
Figure 4.13 also shows that the `MySQL-Req1` virtual machine requirement can also be specified for another blueprint in another Blueprint Model using the BSL IaaS Module (in this case, it is a PaaS blueprint called `MySQL-BP`). Between the `MySQL-Req1` requirement and the `Ubuntu-Off` offering there is a horizontal link indicating that `Ubuntu-Off` is being used to fulfill `MySQL-Req1`. This is an example of composing an IaaS blueprint.

4.2.6 The BSL PaaS Module

This section explains the language constructs provided by the BSL PaaS module for specifying PaaS blueprints. Figure 4.14 presents the BSL PaaS module in relationships with other BSL modules. The BSL PaaS module extends the BSL core module, putting more focus on specifying the product, technology and resource capacity of a PaaS offering or requirement.

A PaaS offering or requirement must be specified with either the property *product* of type *Platform Product* to indicate its product information, or with the property *technology* of type *Platform Technology* to specify its technology information. The BSL PaaS module defines a *Platform Product* with its name, vendor info, version info, release date, and an URI referencing the web resource to get more information about the product. A platform product implements one or more platform technologies. A

Figure 4.14: The BSL PaaS Module in UML

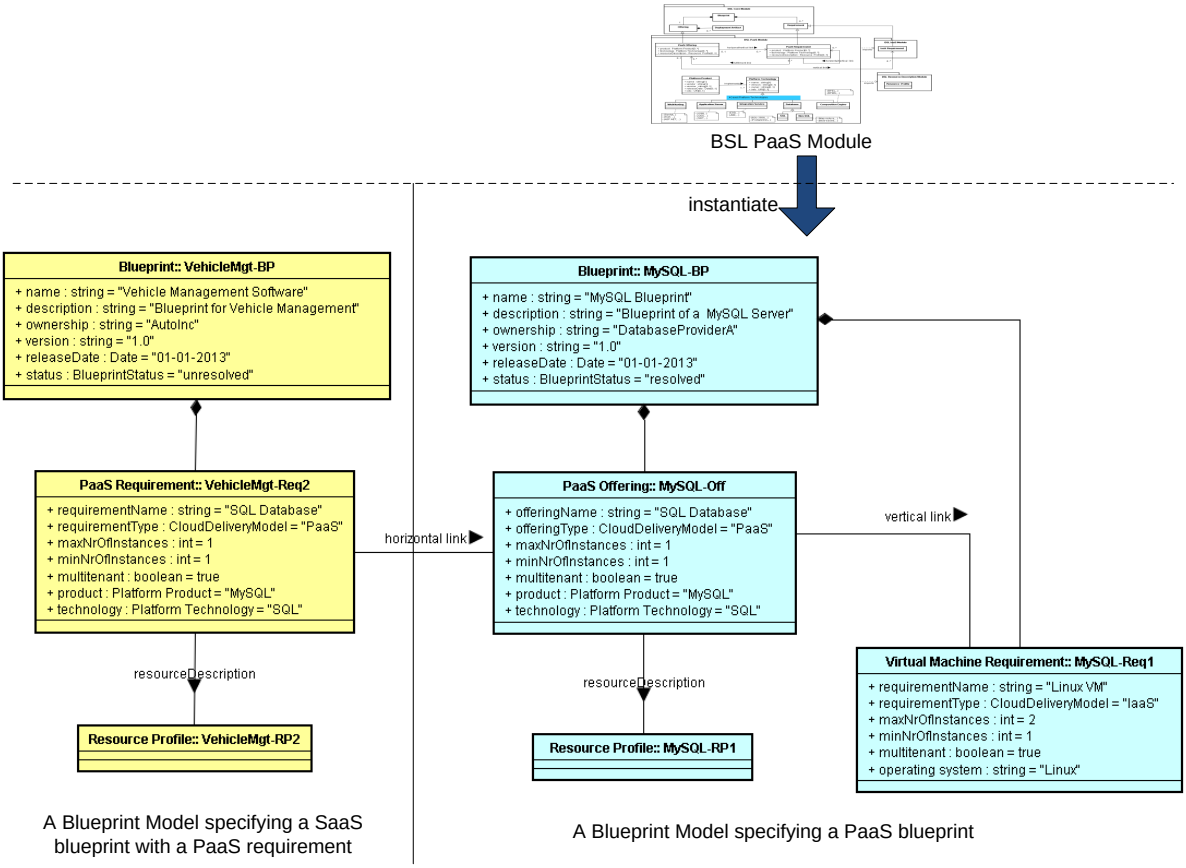


platform technology is specified with a name, version info, owner, and a URI pointing to a web address providing more information on the technology. The BSL PaaS module reuses the 4caast categorization of immigrant platform technologies that should be provided by the PaaSs in the cloud [Binz et al., 2011]:

- Web Hosting: This type of technology is used to implement Web Servers to host web sites. Typical examples are the Servlet Container, PHP, and ASP.NET technologies.
- Application Server: This type of technology is used to implement an Application Server to host applications. Typical examples are the JEE and .Net technology.
- Integration Service: This type of technology is used to implement an EAI component for distributed applications. Typical examples are an Enterprise Service Bus (ESB), or a JMS server.
- Database: Nowadays, database technology can be sub-divided into SQL and non-SQL technologies. Example of non-SQL technologies are MapReduce[5] and

[5]MapReduce: Simplified Data Processing on Large Clusters `http://research.google.com/archive/mapreduce.html`

Figure 4.15: Example of an PaaS blueprint specified by the BSL PaaS Module



BLOB-based technology. Examples of database products that implement MapReduce include MongoDB, Apache Hadoop, etc. Example of implementing BLOB-based is the storage in Microsoft Azure.

- Composition Engine: This type of technology is used to implement service composition engines including the engine for Web Service compositions, e.g. WS-BPEL Engine[6], or for composing telecom services, e.g. the Ericsson composition engine [Niemoeller et al., 2009].

Dependency links between offerings and requirements in a PaaS blueprint may also exist. The BSL PaaS module allows for specifying:

- A horizontal link between a *PaaS Offering* and a *PaaS Requirement* within the same PaaS blueprint. This horizontal link indicates a need for a cross-PaaS integration.
- A vertical link between a *PaaS Offering* and an *IaaS Requirement* within the same

[6]OASIS Web Services Business Process Execution Language (WSBPEL) `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`

PaaS blueprint. This vertical link indicates a need for a vertical PaaS-IaaS deployment.

- A horizontal link between a *PaaS Requirement* and a *PaaS Offering* in two distinctive PaaS blueprints. This fulfillment link indicates that the PaaS offering can be reused to fulfill the PaaS requirement, thus is normally used to compose a PaaS blueprint with another blueprint.

Example of a PaaS blueprint specified by the BSL PaaS Module

By instantiating the language concepts and their relations in the BSL PaaS module, a Blueprint Model can be created for the purpose of specifying an PaaS blueprint. Figure 4.15 depicts a sample Blueprint Model for specifying the `MySQL-BP` blueprint that, according to our running example, is a PaaS blueprint containing the `MySQL-Off` PaaS offering and the `MySQL-Req1` virtual machine requriement. `MySQL` is specified as the platform product of the `MySQL-Off` and `SQL` is specified as its platform technology. The resource capacity of the `MySQL-Off` is specified in the `MySQL-RP1` resource profile. A vertical link is specified between `MySQL-Off` and `MySQL-Req1` to indicate that the PaaS Offering needs to be deployed on a Linux virtual machine.

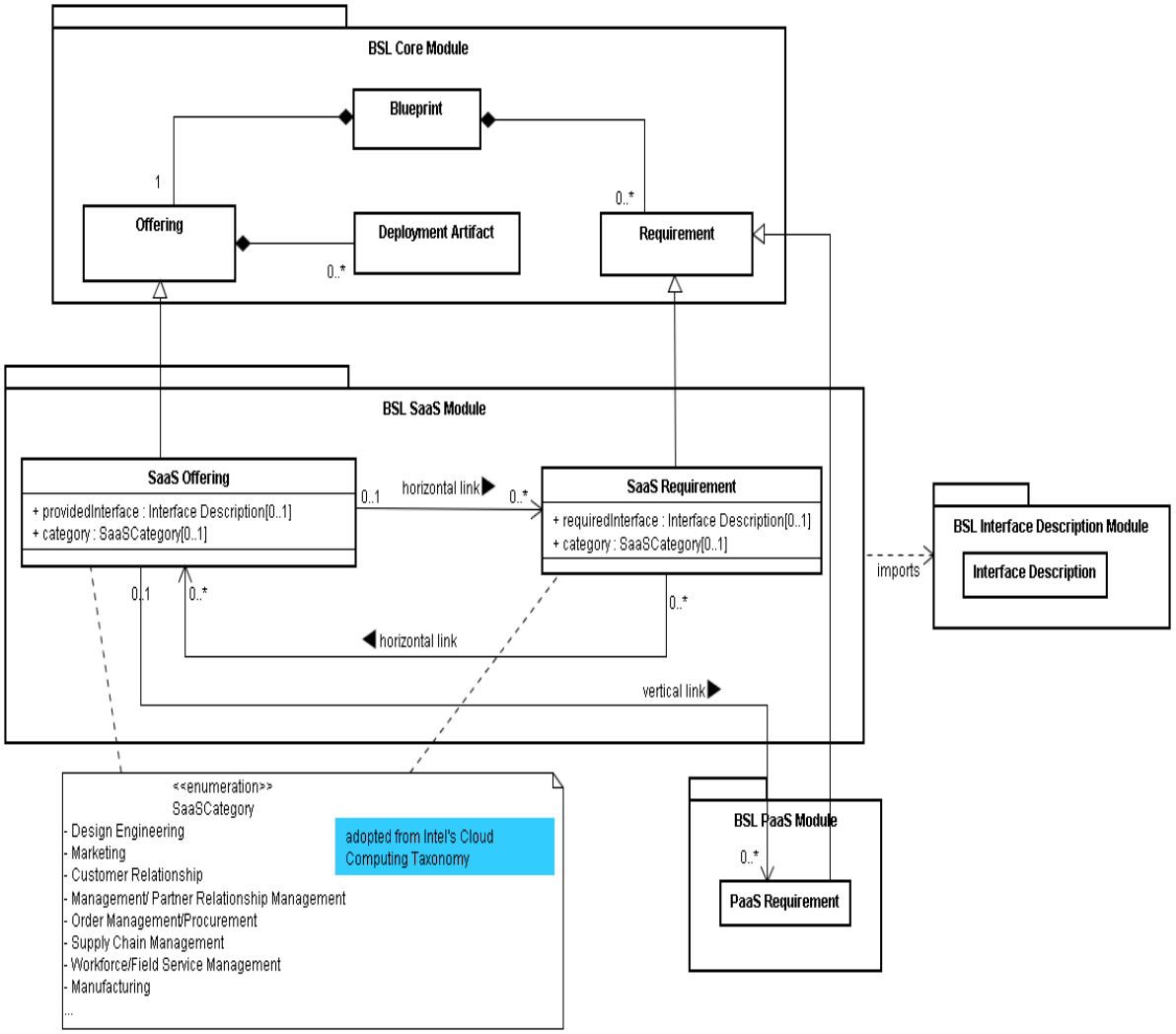
Figure 4.15 also shows that the `VehicleMgt-Req2` PaaS requirement can also be specified for another blueprint in another Blueprint Model using the BSL PaaS Module (in this case, it is the `VehicleMgt-BP` SaaS blueprint). Between the `VehicleMgt-Req2` requirement and the `MySQL-Off` offering there is a horizontal link indicating that `MySQL-Off` is being used to fulfill `VehicleMgt-Req2`. This is an example of composing a PaaS blueprint with another blueprint.

4.2.7 The BSL SaaS Module

This section explains the language constructs provided by the BSL SaaS module to describe SaaS blueprints. Figure 4.16 presents the BSL SaaS module in relationships with other BSL modules. In particular, the BSL SaaS module extends the BSL core module by providing new language concepts for defining a SaaS offering, SaaS deployment artefact, and SaaS requirement. A *SaaS Offering* has all the properties of an *Offering*, and may further provide an *Interface Description* to enable programmatic interactions with the consumers. As we will see later in the discussion about the *Interface Description* of a SaaS in Section 4.2.4, existing languages or specification schemas can be reused to describe these the two levels of a SaaS interface description: the signature and protocol. Similarly, a *SaaS Requirement* has also all the properties of a *Requirement*, and may further expose a required *Interface Description*.

Another important property of a SaaS offering or requirement is its *category*. This property allows for a better matchmaking between SaaS offerings and requirements, i.e. by first matching their categories. To the best of our knowledge, there have been not many initiatives in categorizing SaaS or, in general, services on the Web.

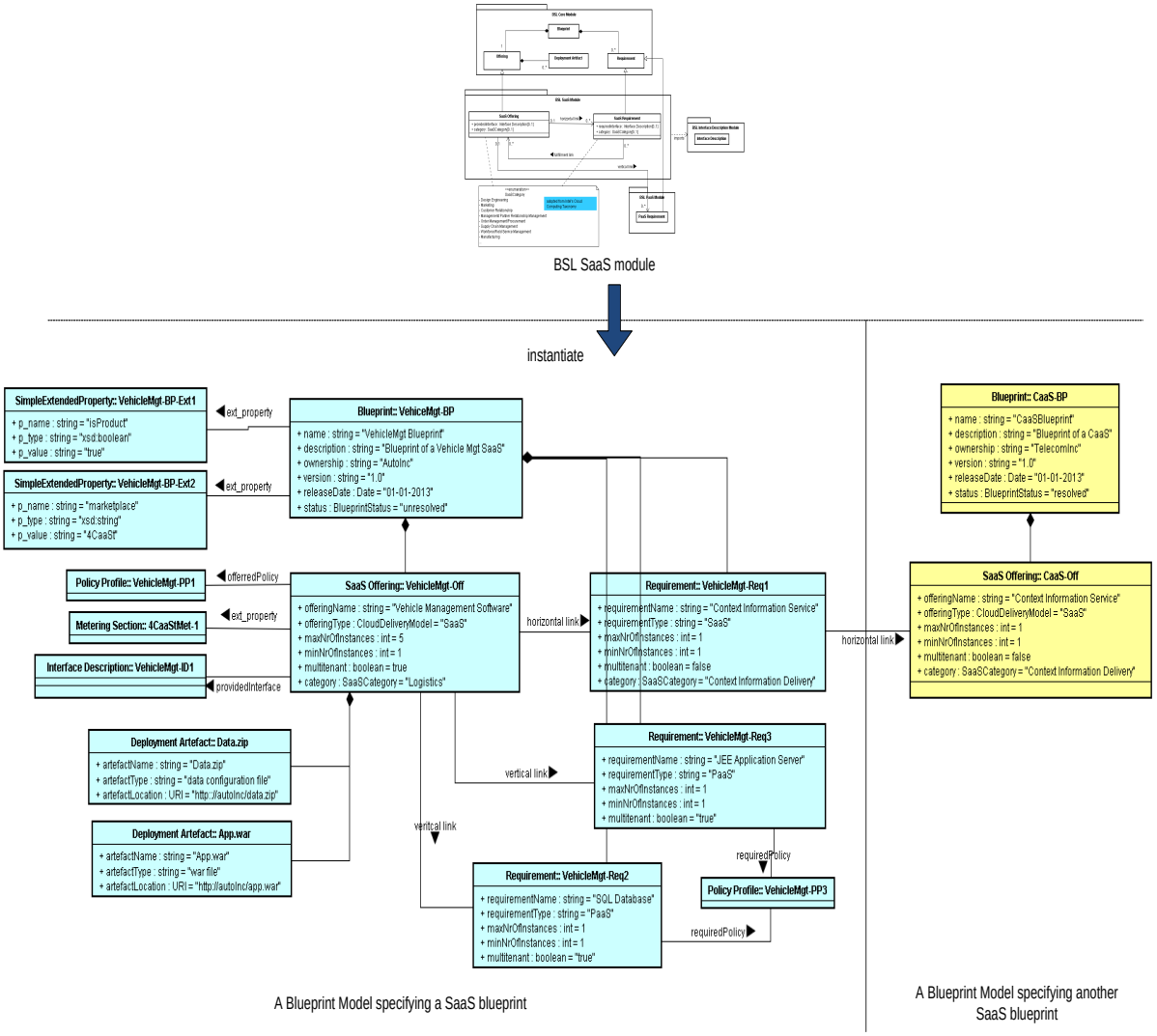
Figure 4.16: The BSL SaaS Module in UML



The BSL SaaS module reuses a comprehensive SaaS categorization provided by Intel [Intel, 2010] for defining the value range of the property *category*. The BSL SaaS module has also defined the relationships among its language constructs to allow for the specification of the vertical, horizontal, and fulfillment link. Notably are the following blueprint dependency links defined by the BSL SaaS module:

- *Horizontal links* could exist from a SaaS offering to many SaaS requirements to indicate functional dependencies on another third-party SaaS.
- *Horizontal links* could exit from a SaaS requirement to a SaaS offering of two distinctive blueprints to indicate that the SaaS requirement can be fulfilled by the SaaS offering. Obviously, a SaaS requirement can be fulfilled by several different SaaS offerings.
- *Vertical links* could exist between a SaaS offering and many PaaS requirements

Figure 4.17: Example of a SaaS blueprint specified by the BSL SaaS Module



(defined by the BSL PaaS module) to indicate the deployment dependencies on the third-party PaaSs.

Finally, it is worth to remark that a common practice on the SaaS layer is to reuse existing standards for specifying business-related meta-data of a SaaS. The Universal Service Description Language (USDL) and the SOAML languages are the most prominent candidates for the reuse. A business meta-data specification described in one of these 2 languages can be easily incorporated into a SaaS blueprint specification by using the extended properties *ext_property* provided by the BSL Core module in Section 4.2.1.

Example of an SaaS blueprint specified by the BSL SaaS Module

By instantiating the language concepts and their relations in the BSL SaaS module, a Blueprint Model can be created for the purpose of specifying a SaaS blueprint. In Figure 4.6, we have shown a sample Blueprint Model containg the `VehicleMgt-BP` blueprint. The offing of this blueprint is, in fact, classified as a SaaS offering. Hence, using the BSL SaaS module, Figure 4.17 extends the specification of the `VehicleMgt-BP` blueprint with more SaaS-related meta-data information. For instance, the `VehicleMgt-Off` offering is classified as a SaaS offering in the `Logistics` industry domain. Details of the technical interface of this offering has been specified in the `VehicleMgt-ID1` interface description, which has been specified using the BSL Interface Description Module.

Between the offering `VehicleMgt-Off` and the requirement `VehicleMgt-Req1`, a horizontal link has been specified to indicate a functional dependency from the SaaS offering to a SaaS requirement. Vertical links have been specified to indicate the deployment dependencies between the SaaS offering `VehicleMgt-Off` and the two PaaS requirements `VehicleMgt-Req2` and `VehicleMgt-Req3`.

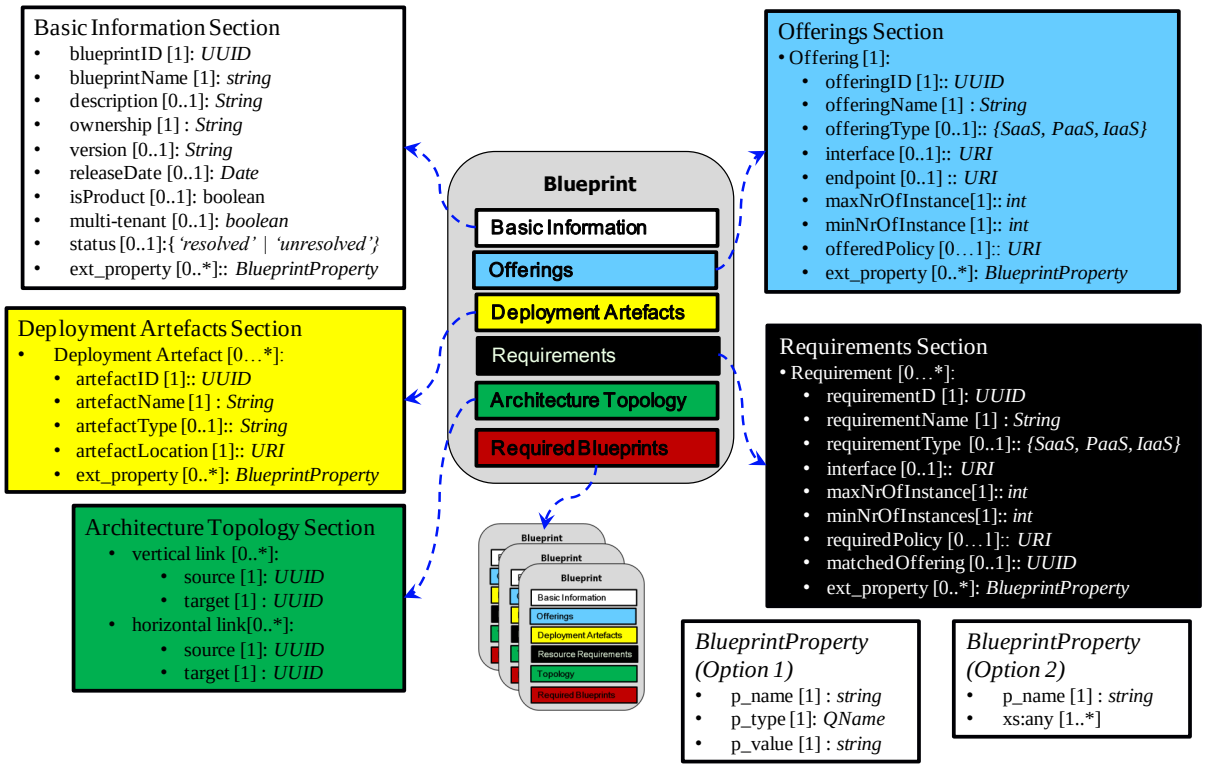
Lastly, Figure 4.17 also shows that the SaaS requirement `VehicleMgt-Req1` has a horizontal link to the `CaaS-Off` offering of another blueprint. This type of horizontal link (from a requirement to an offering of two distinct blueprint) indicates a fulfillment dependency. A horizontal link in this case signifies an example of composing two SaaS blueprints.

4.3 BSL Concrete Syntax in XML

The BSL model introduced in the previous Section 4.2 is an abstract syntax model for specifying blueprints, without proposing any concrete representation techniques for the users. In the context of cloud computing where blueprints, as the cloud service specifications, are supposed to be exchanged frequently on the Web for the purpose of combining cloud services across different providers, a semi-structured data format seems appropriate for the purpose of representing blueprints. Towards this goal, this section proposes a XML schema template for describing blueprints in XML documents.

Inclined with the emergence of the web, the concept of semi-structured data emerged in the late 20th century to target the desire for data exchange and integration across heterogeneous data sources on the Web. Semi-structured data are those defined with irregular, often unknown-in-advance schema definition, or even when the schema is known, it may change often and without notice [Suciu, 1998]. Semi-structured data models are hence called self-describing as the schema definition is embedded in the data model so that the data can be read, parsed, and understood easily. Among others, XML is the most appealing technique used for describing semi-structured data. XML is a markup language for describing data in a document that can be easily understood and exchanged on the Web. To define the meta-structure of a type of XML documents, one has to define a document called XML Schema. Among others, the XSD, which was published as a W3C recommendation in May 2001, is one

Figure 4.18: The Blueprint XSD Template



of the most used languages for defining an XML schema document[7].

Based on the BSL model in Section 4.2, we have developed and proposed in [Nguyen et al., 2011][Nguyen et al., 2012a] the *Blueprint XSD Template* as the BSL concrete syntax for specifying blueprints in XML documents. Within the 4caaSt community [European Comission, 2010], we are extensively using the blueprint XSD template as the uniform specification and interchange format for blueprints. Blueprints described using this template can be easily queried from a marketplace repository, and then customized and composed by the CSBA engineers to address their application requirements.

The Blueprint XSD template is depicted in Figure 4.18. The BSL Syntactic Mapping from the BSL model to the template is quite straightforwards. Each concept in the BSL model is mapped to an XSD element in the template. Properties of a model concept are mapped to properties of the corresponding XSD element. In addition, an XSD element representing a blueprint, offering, deployment artefact, or requirement, is specified with a Universally Unique Identifier (UUID)[8] so that it can be uniquely referenced within a blueprint and across blueprints. There are some specific details of the mapping that are listed in the following:

[7] http://www.dblab.ntua.gr/~bikakis/XML%20and%20Semantic%20Web%20W3C%20Standards%20Timeline-History.pdf

[8] http://en.wikipedia.org/wiki/Universally_unique_identifier

- The blueprint XSD template does not distinguish between SaaS, PaaS, and IaaS blueprints. The classification of a blueprint can be determined by checking the field *offeringType* in the Offering Section.

- The "horizonal link" between the concepts "Requirement" and "Offering" in the BSL model is mapped into the property *matched_offering* of the XSD element "Requirement". The *matched_offering* property specifies a UUID reference to the offering that can fulfill the requirement. Therefore, a blueprint is "resolved" if all the *matched_offering* property of all the requirements are specified with a value.

- In case the template is used to specify a "resolved" blueprint, the "Required Blueprints" section is used to store the blueprints whose offerings have been identified to fulfill a requirement. In case of specifying an "unresolved" blueprint, this section is empty.

- The vertical and horizontal links within a blueprint are specified in a separate template section "Topology Section", each with a "source" and a "target" property that specify the UUID references to the source and target elements of the link.

- The BSL Interface Description module and the BSL Policy Description module have not been mapped into the blueprint XSD template. Instead, a simplified approach has been used in the blueprint XSD template that uses a *URI* to reference an Interface Description or a Policy Profile, with the assumption that this document has been specified by a well-known standard specification format, e.g. WSDL and WS-Policy.

- The BSL Resource Description module has not been mapped into the blueprint XSD template. Instead, a blueprint provider can use the *ext_property* in each template section to specify the resource properties.

Please note that the blueprint XSD template can also be customized for specifying blueprints in a particular category. Similar to the extended properties of the concepts in the BSL model, the blueprint XSD template supports defining user-defined properties *ext_property* for the "Blueprint, Offering, Deployment Artifact and Requirement" XSD elements. Each *ext_property* element may be needed for a particular category of cloud services and thus can be specified with a property name *p_name*, property type *p_type*, and property value *p_value*, or with a property name *p_name* and an XML wildcard *xs:any*.

4.4 Formalizing the BSL Semantics

The previous Section 4.3 introduces a blueprint XSD template as a concrete XML-based syntax of the BSL. This blueprint XSD template allows cloud service providers to describe their blueprints in XML documents. However, XML is barely a syntax format for information exchange on the Web, i.e. an informal XML representation of a blueprint does not provide an explicit specification of the intended meanings of the blueprints, their elements and their links. There is a need to formalize the semantics of the BSL to provide a precise meanings of the blueprints, their elements and their links. The purpose of defining the formal semantics of the BSL is twofold:

- To support a more precise blueprint discovery and selection process: Given the formal semantics, blueprints can be more precisely matched, compared and selected.
- To support an automatic composition of blueprints: A formal semantics of blueprint will help overcome the interoperability issue between the blueprints.

Given this motivation, we will first explain in Section 4.4.1 the choice of the Web Ontology Language (OWL) as the knowledge representation language for formalizing the semantics of the BSL. The formalization has been conducted as the transformation of the BSL model into a set of inter-related OWL models. The result of the transformation will be presented in Section 4.4.2. The limitations of our transformation will also be reported. Lastly, by using a technique provided by the World Wide Web (W3C) consortium for serializing an OWL model into an XML document [Motik et al., 2009], we also present in Section 4.4.2 a concrete OWL/XML representation format for the BSL. In comparison with the purely syntactic XML-based representation technique presented in Section 4.3, the OWL/XML representation technique is enriched with more semantics to provide a more precise and consistent way of representing and linking blueprints as the resources and resource relationships on the Internet.

4.4.1 The choice of Web Ontology Language (OWL) for formalizing the BSL Semantics

The history of semantic web started with the invention of the Resource Description Framework (RDF) [Manola & Miller, 2004], which is an approach to enhance the lack of semantics of XML by providing a consistent, standardized way to describe resources on the Web and the relationships among the resources. The RDF Schema (RDFS) [Brickley & Guha, 2004] extends RDF by providing mechanisms to describe a group of related resources as a class and the relationships between the classes as properties. A class defined by RDFS may have multiple sub- and superclasses and a property in RDFS can be defined with multiple pairs of domain and range classes.

The Web Ontology Language (OWL) [McGuinness & van Harmelen (Eds.), 2004] is an ontology modeling language recommended by the W3C that adds more vocabulary to describe classes and properties of the RDFS, e.g. it can describe relations between classes (such as disjointness), cardinality (for example, "exactly one"), equality, richer typing of properties, and characteristics of properties (such as symmetry). OWL is designed for use by applications that need to process the content of information rather than just presenting information to humans [Balani, 2005]. It facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDFS by providing additional vocabulary along with formal semantics [Balani, 2005]. It has been claimed by Horrocks in [Horrocks et al., 2007] that OWL is based on the Description Logic, which is "a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way" [Baader et al., 2008]. In fact, the formal semantics of OWL (and all its sub-languages) has been provided by Horrocks in [Horrocks et al., 2007] using SHOIN(DL) and SHIF(DL), the two very expressive members of the Description Logic(DL) family.

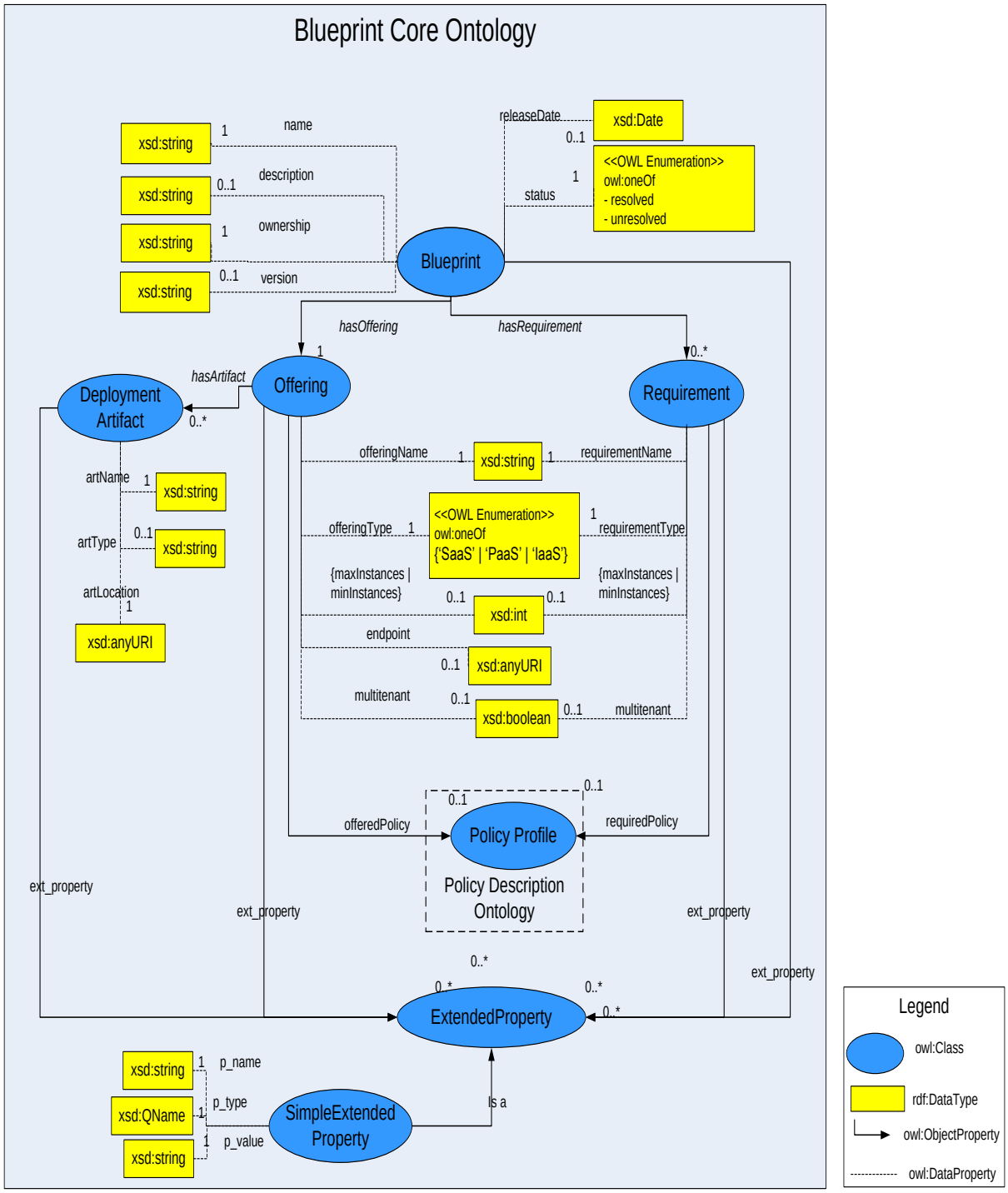
Given that OWL is formally underpinned by the Description Logic, which is a family of formal knowledge representation language [Baader et al., 2008], the transformation of the BSL model into a set of inter-related OWL models would provide the sufficient level of formalism for our BSL contribution. This motivation has led us to the choice of OWL to formalize our BSL model. The result of the BSL-to-OWL transformation will be reported in the next Section 4.4.2. Furthermore, the W3C has also provided a concrete XML-based syntax called OWL/XML [Motik et al., 2009] for serializing an OWL model into an XML document for information exchange purpose. We have also used OWL/XML as the concrete representation technique for specifying blueprints in XML documents. In the next section, the OWL/XML representation of the BSL will also be reported.

4.4.2 BSL-to-OWL Transformation

Having modeled the BSL model in the previous Section 4.2 using the UML class diagram, the contribution of this section is to formalize the BSL semantics by transforming the BSL model into a set of inter-related OWL models: the *Blueprint Core Ontology* capturing the formal semantics of the BSL core module, the *SaaS/PaaS/IaaS Blueprint Ontology* capturing the formal semantics of the BSL SaaS/PaaS/IaaS modules, the *Interface Description Ontology* capturing the formal semantics of the BSL interface description module, the *Resource Description Ontology* capturing the formal semantics of the BSL resource description module, and the *Policy Description Ontology* capturing the formal semantics of the BSL policy description module.

The most precise way of transforming a UML model to an OWL model is to use

Figure 4.19: Blueprint Core Ontology



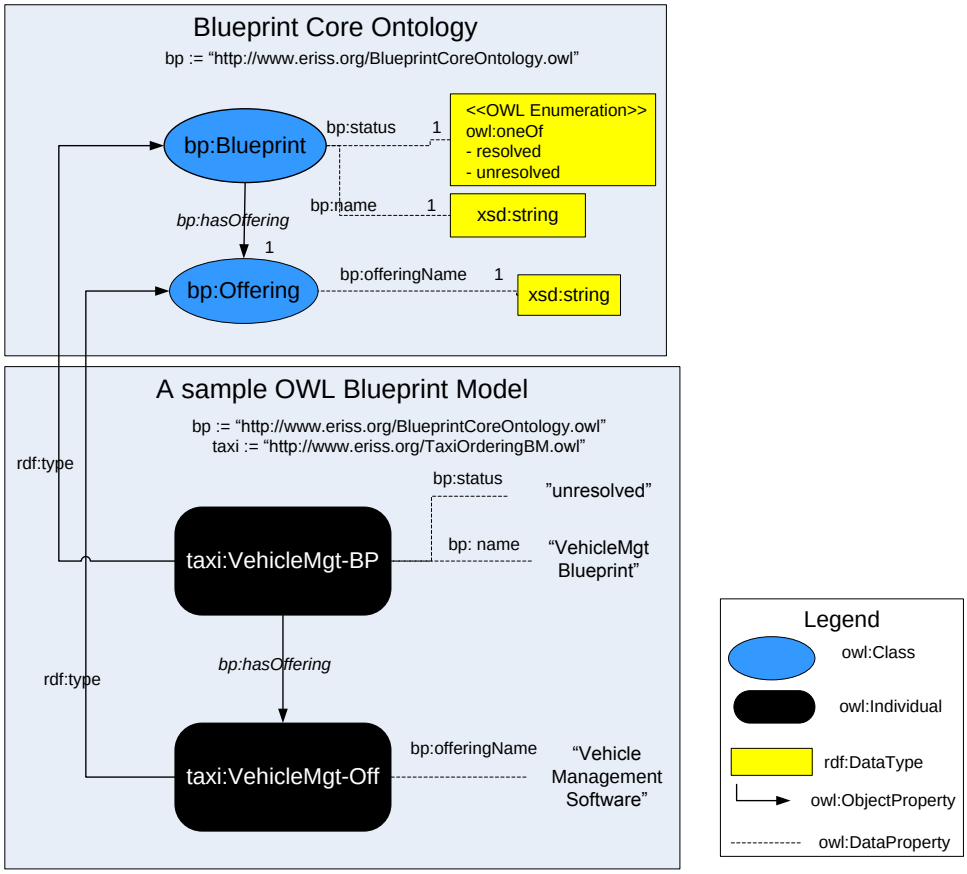
the Ontology Definition Metamodel (ODM) defined by the OMG [OMG, 2009]. The ODM document provides a UML profile for modelling RDF and OWL. It also provides a full guideline for the UML-to-OWL transformation. However for brevity reason, we decide for a more light-weight and ad-hoc approach by following the transformation guidelines in [Hart et al., 2004][Gasevic et al., 2004] to transform the BSL modules

into OWL models. As an example, Figure 4.19 illustrates the *Blueprint Core Ontology* described in OWL as the result of transforming the BSL core module. For brevity reason in this section, we do not discuss the OWL models of other BSL modules in this section. In the following, the transformation techniques are summarized:

- Each UML class is mapped to an OWL class.
- A relation between two UML classes is mapped to an object property between two corresponding OWL classes. An object property in OWL is defined between two OWL classes, one of which is the domain and the other is the range.
- Attributes of an UML class with simple data types are mapped to data properties of the corresponding OWL class. A data property in OWL uses the data types defined by the XML schema, e.g. xsd:string, xsd:int, etc., as the specification of the data range.
- Attributes of an UML class with enumeration data type are mapped to OWL enumeration data types.
- Cardinality constraints between two UML classes or between a UML class and its attributes are preserved for OWL object properties or data properties.
- Composition relations are used in the UML model indicate the part-of integral relationships, e.g. an offering or requirement only belongs to a blueprint and if the blueprint ceases to exist, all of its offering and requirements cease to exist too. As pointed out in [Hart et al., 2004], composition is not a supported modeling feature in OWL. OWL allows only for declaring an object property or data property as "functional", i.e. an instance of the property domain has only one unique instance of the property range, yet it supports no existence constraint between the instances of the classes. Hence, existence constraints are left out of the mapping and defined as additional constraints on top of the OWL blueprint schema.
- The *SimpleExtendedProperty* concept in the BSL core module has been proposed as a simple syntax for specifying a user-defined property for a language concept in the BSL core module. The Blueprint Ontology has been developed as the formalization of the BSL core module semantics. Hence, an OWL class `SimpleExtendedProperty` has been introduced for the OWL blueprint schema that represents a user-defined property with a name, type, and value.

Our intention for doing the BSL-to-OWL transformation was only to provide a formal underpinning of our BSL contribution. We did not aim to provide a rich semantic model for the blueprints. Therefore while doing the transformation, we did not use

Figure 4.20: A Sample OWL Blueprint Model containing the `VehicleMgt-BP` blueprint



the full features of OWL that were originally developed for the semantic web community. The **limitation of our transformation** is explained in the following in terms of a confined set of selected OWL features:

- We use the basic features to define an OWL Class, Individual, Data Type, Object Property, and Data Property.
- We use the subClassOf and disjointWith axioms for the Classes.
- We use the cardinality and existence restrictions on the Object and Data Properties.
- We classify whether an Object or Data Property is functional or not.

Despite the aforementioned limitation, our proposed OWL models for the BSL already capture all the necessary formal knowledge for representing meta-data of a cloud service. Using these OWL models, instances of the OWL classes (called OWL individuals) can be populated to represent blueprints and their associated meta-data elements. To avoid the confusion between an OWL class and an OWL individual, the populated

OWL individuals are defined in a separate *OWL Blueprint Model*. Figure 4.20 illustrates a sample OWL Blueprint Model containing two sample OWL individuals, i.e. the `VehicleMgt-BP` blueprint and `VehicleMgt-Off` offering in our running example, which have been populated from the OWL classes "Blueprint" and "Offering" of the Blueprint Core Ontology. Please note that for brevity reason, Figure 4.20 does not show the complete `VehicleMgt-BP` blueprint in the sample OWL Blueprint Model, i.e. not all the data properties of the individuals are shown in the example and further individuals specifying requirements, deployment artefacts, extended properties, interface description, resource profiles, and policy profiles are absent.

All the OWL models described in this section have been modeled using the Protege tool[9]. Based on the OWL/XML syntax proposed in [Motik et al., 2009], Protege supports also the serialization of the OWL models into XML documents (we call them OWL/XML documents to distinguish from the purely syntactic XML representation in Section 4.3). The sample OWL Blueprint Models have also been created and serialized by Protege as OWL/XML documents capturing the sample blueprints.

[9]http://protege.stanford.edu/

CHAPTER 5

BLUEPRINT MANIPULATION TECHNIQUES

The blueprint approach aims to compartmentalize the monolithic cloud service delivery stack in three different layers of cloud services, i.e. SaaS, PaaS, IaaS, and use the concept of blueprints as the uniform specification of cloud services across all these three layers. We have proposed in the previous chapter 4 a Blueprint Specification Language (BSL) for cloud service providers to specify their blueprints in a uniform and consistent manner. The advantage of following the blueprint approach is that blueprints specified by the BSL can be flexibly manipulated and composed into an end-to-end CSBA configuration[1]. The *Blueprint Manipulation Techniques* (BMTs) have been developed exactly for this purpose. Inspired by the model management operators that have been developed in [Melnik, 2004] for the purpose of manipulating data models, the BMTs have been developed as a set of *BMT Operators* to support the CSBA engineers with the publishing, querying, and composition of several blueprints available in a repository.

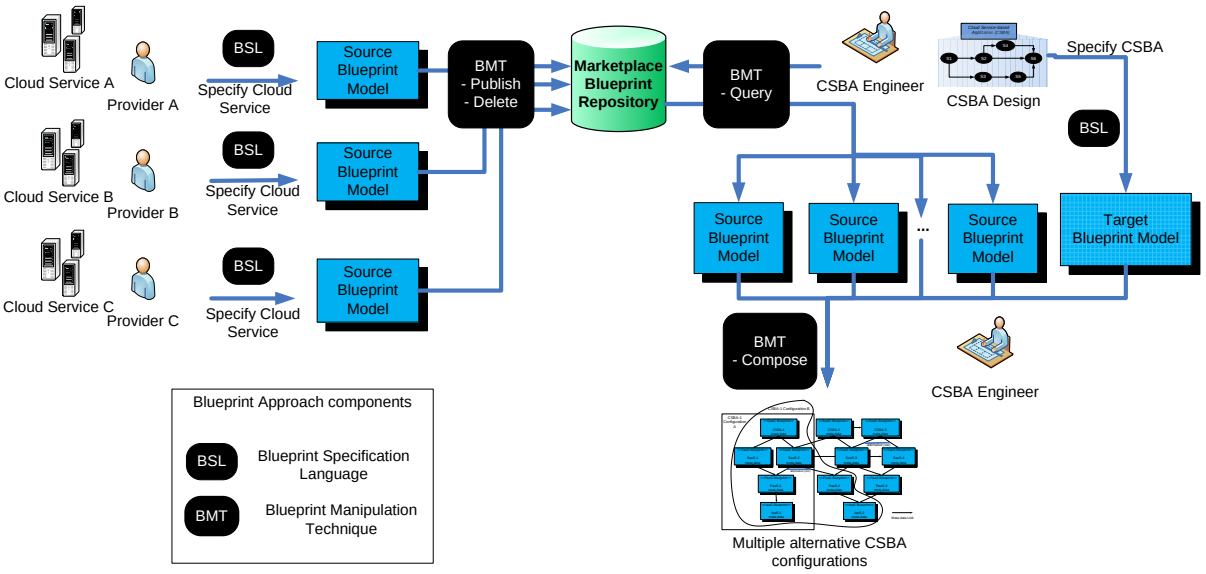
5.1 Introduction

The BMTs are defined as the techniques that support the publishing, deleting, querying, and composition of blueprints. The purpose of this section is to give a general introduction of the BMTs by explaining how the BMTs can be used to support the CSBA engineers and cloud service providers within the CSBA engineering lifecycle.

In Figure 5.1, we review the CSBA Engineering Lifecycle that has been introduced in the previous section 1.2. By using the BSL, a blueprint can be specified in a *Blueprint*

[1]According to Definition 1.3, Section 1.2, an CSBA configuration is a composition of blueprints required for configuring the deployment environment of an CSBA

Figure 5.1: BMT Supports for the CSBA Engineering Lifecycle



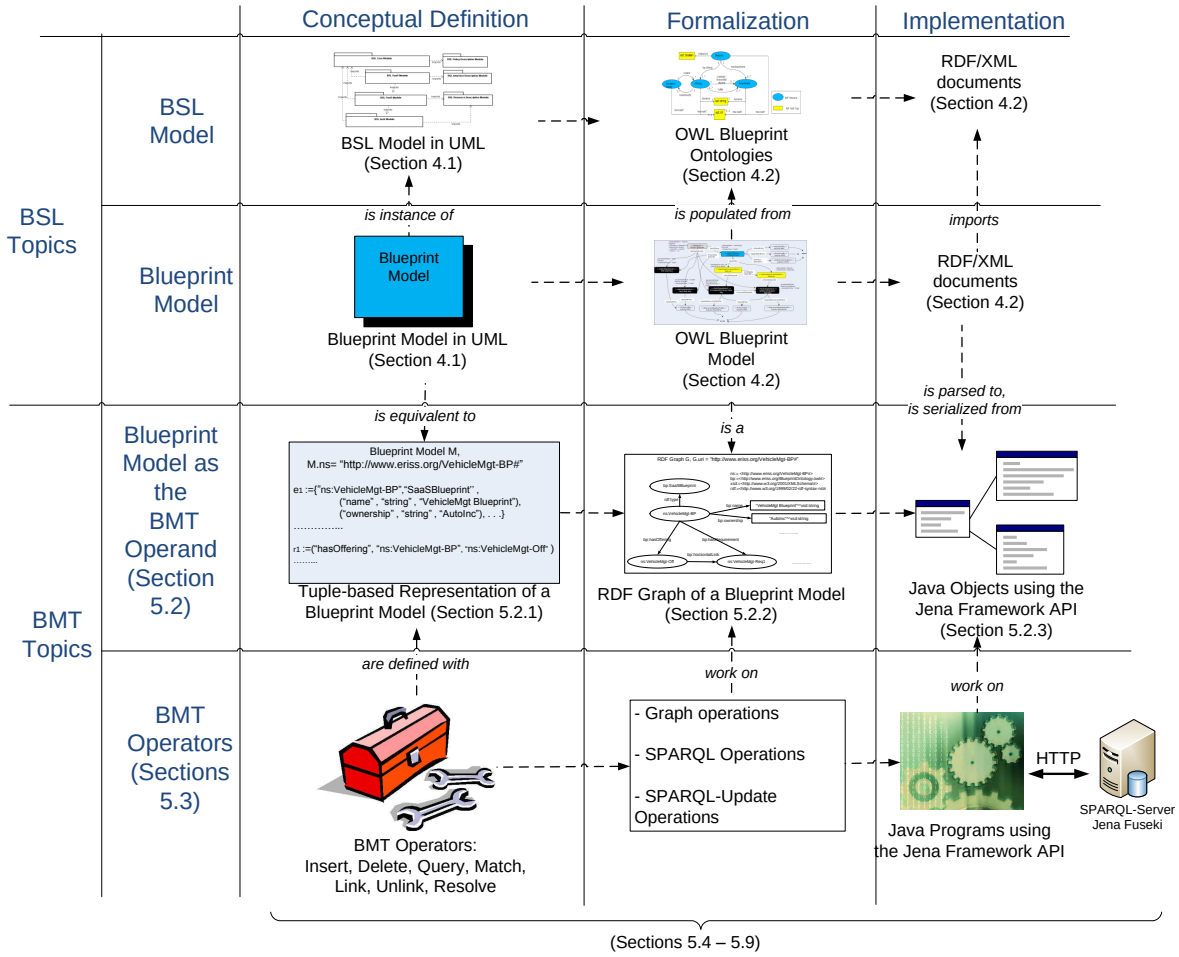
Model, which is an instance of the BSL model. We distinguish between a source blueprint models specifying existing blueprints in a repository and a target blueprint model specifying a blueprint under development[2]. The role of each BMT technique is also shown in Figure 5.1. In particular:

- Supports for a Cloud Service Provider:
  - ***Publish***: A cloud service provider can use this technique to publish a source blueprint model to a blueprint repository. A blueprint repository, for instance, may belong to a marketplace that enables the advertisement and purchasing of blueprints.
  - ***Delete***: A cloud service provider can use this technique to delete a source blueprint model from a blueprint repository.
- Supports for a CSBA engineer:
  - ***Query***: An CSBA engineer can use this technique to query for the needed source blueprint models from a repository.
  - ***Compose***: An CSBA engineer can use this technique to compose two or more blueprint models to form an CSBA configuration.

The aforementioned BMT techniques have been developed as a set of *BMT Operators* that support the manipulation and composition of blueprint models. Hence, the

[2]The classification between source and target blueprints has been introduced in the previous Section 3.2.5

Figure 5.2: The BMT Topics in relation with the BSL Topics



blueprint models are considered as the *BMT Operands*. Figure 5.2 introduces the topics that will be covered in this chapter:

- Blueprint models as the BMT Operands: A blueprint model is an instance of the BSL model that specifies a blueprint. It serves as the operand of a BMT operator. In Section 5.2 we introduce the concept of a blueprint model through the following topics:
  - Conceptual Definition: Section 5.2.1 introduces a tuple-based representation of a blueprint model. This representation is equivalent to the UML-based representation that has been introduced in the previous Section 4.2, yet is simpler for being used to explain the input and output of a BMT operator.
  - Formalization: Section 5.2.2 formalizes a blueprint model as a Resource Description Framework (RDF) Graph. In the previous Section 4.4, a blueprint model has been introduced as a model described by the Web Ontology Language (OWL). Since the OWL language extends the RDF vocabulary, an

OWL model is also a RDF graph. Formalizing a blueprint model as a RDF Graph has the advantage that the BMT operators that work on the blueprint models can be formalized as the basic graph operations, the SPARQL operations [W3C, 2008] for querying RDF Graphs, or the SPARQL-Update operations [W3C, 2012] for updating RDF Graphs.

  - Implementation: Section 5.2.3 introduces the implementation of a blueprint model as a set of Java objects implementing the Jena Framework API[3]. It is also possible to use the Jena Framework API to parse a RDF/XML document into a blueprint model, and vice versa to serialize a blueprint model into a RDF/XML document.

- BMT Operators: Section 5.3 presents an overview of all the BMT operators. Then, in the subsequent sections 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9, each BMT operator is introduced in detail with the following topics:

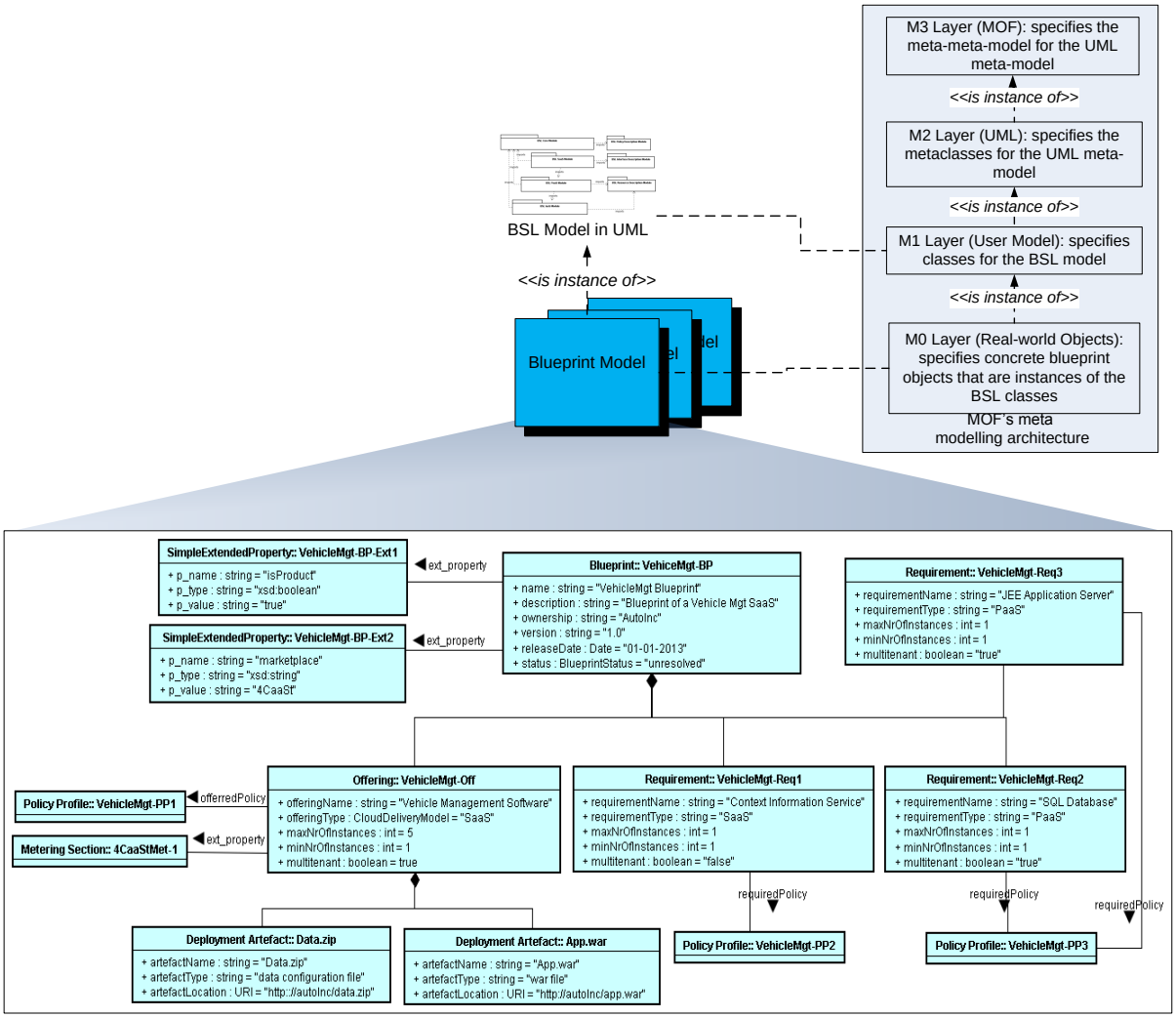
  - Conceptual Definition: the signature and functionality of each operator will be introduced with the blueprint models as its operands.
  - Formalization: Since a blueprint model is formalized as a RDF Graph, the formalization of a BMT operator follows one of the following techniques:
    * By using basic graph operations: Basic graph operations can be used for modifying an RDF graph or identifying matchings between two RDF graphs.
    * By using SPARQL operations: The SPARQL operations [W3C, 2008] can be used for querying RDF Graphs from a RDF Dataset. The SPARQL language defines a RDF Dataset as a collection of RDF Graphs, against which a SPARQL query can be executed.
    * By using SPARQL-Update operations: The SPARQL-Update operations [W3C, 2012] can be used for updating and managing RDF Graphs in a RDF Graph Store. The SPARQL-Update language defines a RDF Graph store as a similar concept to a RDF Dataset[4]. However unlike the RDF Dataset, a RDF Graph Store also allows for adding and deleting RDF graphs.
  - Implementation: The BMT operators have been implemented as Java programs using the Jena Framework API. The Jena Framework API supports the creation and manipulation of RDF Graphs, as well as the execution of SPARQL queries and SPARQL-Update statements against a RDF Graph Store. The Jena Fuseki server[5] is used as a RDF Graph Store server.

[3]Jena Framework API: `http://jena.apache.org/`

[4]We use the term RDF Graph Store synonymously for a RDF Dataset

[5]Jena Fuseki: `http://jena.apache.org/documentation/serving_data/`

Figure 5.3: A sample Blueprint Model in UML



A sample Blueprint Model in UML specifying the VehicleMgt-BP blueprint

5.2 Blueprint Model as the BMT Operand

During the introduction of the BSL model in the previous Section 4.2, we have also introduced the concept of a *blueprint model* as an instance of the BSL model that is created to specify a blueprint. Examples of blueprint models have been introduced in the previous Section 4.2 using the UML class diagram notations to specify the sample instantiated blueprints.

A blueprint model is defined as an instance of the BSL model that specifies a single blueprint. It contains model elements that are instances of the BSL classes, e.g. the instances of the classes "Blueprint", "Offering", "PolicyProfile", etc. Figure 5.3 illustrates an example of a blueprint model specifying the `VehicleMgt-BP` blueprint in the *Taxi Tilburg Scenario*. The blueprint model in this example is represented using the UML class diagram notations. It is positioned on the M0 layer of the MOF's meta-modeling

architecture, which is the layer of real-world object model.

Although a blueprint model can be represented using the UML class diagram notations like the examples introduced in the previous Section 4.2, a graphical UML-based representation of a blueprint model is not suitable for defining the operators that work on it. We introduce in the following an abstract tuple-based representation of a blueprint model that is equivalent to the UML-based representation, yet simpler for explaining the input and the output of each operator that we are going to define in the subsequent sections.

5.2.1 A Tuple-based Representation of a Blueprint Model

In this section, we introduce a tuple-based representation for a blueprint model that is equivalent to its UML-based graphical representation. The reason for introducing this tuple-based representation is because it is suitable for defining inputs and outputs of the BMT operators that we will define in the subsequent sections.

An element e in a Blueprint Model M, i.e. $e \in M$, is an instance of a BSL class, e.g. "Blueprint", "Offering", "Requirement", etc. It is represented as the following tuple:

$$e := \{id :: URI,\ \ class :: string,\ \ att^*\}, \tag{5.1}$$

where

- id: the unique ID of an element e defined in terms of an URI reference.
- $class$: the name of the BSL class from which e is instantiated, e.g. "Blueprint", "Offering", etc.
- att^*: zero or more attributes. Each attribute att is defined with a 3-tuple of attribute name, type, and value:

  $att := (name :: string,\ \ type :: string,\ \ value :: string|URI)$ where

  - $name$: the name of the attribute as defined in the BSL class, e.g "name", "type", "maxNrOfInstances", etc.
  - $type$: the type of the attribute as defined in the BSL class. It could be a primitive type, e.g. string, int, boolean, etc., or a complex type defined by another BSL class.
  - $value$: the value of the attribute.

    If the $type$ is specified as a primitive type, then $value$ should conform to the $type$. For instance, $type$ = "int" & $value$ = "6", or $type$ = "$boolean$" & $value$ = "$true$".

    If the $type$ is specified as a complex type defined by a BSL class c, then $value$ stores the URI reference to another element e' that is an instance of the class c. For instance, $type$ = "$PolicyProfile$" & $value$ = "ns:VehicleMgt-PP1".

A relation may exist between two elements in a blueprint model. These relations have been defined in the BSL model as the relations between the BSL classes, e.g. "hasOffering", "hasRequirement", "horizontalLink", "verticalLink", etc. A relation r in a Blueprint Model M, i.e. $r \in M$, between the source element e_s and the target element e_t is represented as the following tuple:

$$r := (name :: string,\ \ id_s :: URI,\ \ id_t :: URI), \tag{5.2}$$

where

- *name*: the name of the relation as defined by the BSL, e.g. "has Offering", "has Requirement", "horizontal link", "vertical link", etc.
- id_s: the unique ID of the source element e_s in M, i.e. $\exists! e_s \in M,\ r.id_s = e_s.id$.
- id_t: the unique ID of the target element e_t, i.e. $\exists! e_t,\ r.id_t = e_t.id$. Please note that e_t may belong to another Blueprint Model M'.

Lastly, a Blueprint Model M is defined with a namespace ns, a finite set of elements e_i, and a finite set of relations r_j:

$$M := \{ns :: string,\ \ e^*,\ \ r^*\}, \tag{5.3}$$

where

- ns: the namespace of the model M. It is used as the unique ID of the blueprint model.
- e^* the elements in M.
- r^* the relations in M. Please note that the source element of a relation has to belong to M, i.e. $\forall r_j \in M,\ \exists! e_i \in M,\ r_j.id_s = e_i.id$, but the target element can belong to another Blueprint Model M'.

Example 5.1 (Example of a Tuple-based Representation of a Blueprint Model) *We introduce in the following a sample blueprint model M specifying the* VehicleMgt-BP *blueprint in the* Taxi Tilburg Scenario*:*

URI Prefix declarations:

$$\begin{aligned} ns &:= \text{“http://www.eriss.org/VehicleMgt-BP\#''} \\ ns1 &:= \text{“http://www.eriss.org/CaaS-BP\#''} \\ ns2 &:= \text{“http://www.eriss.org/MySQL-BP\#''} \\ ns3 &:= \text{“http://www.eriss.org/PostgreSQL-BP\#''} \\ ns4 &:= \text{“http://www.eriss.org/JBoss-BP\#''} \end{aligned} \tag{5.4}$$

Elements in M:

$$\begin{aligned} e_1 := \{&\text{“ns:VehicleMgt-BP''}, \text{“}SaaSBlueprint\text{''}, \\ &(\text{“}name\text{''}, \text{“}string\text{''}, \text{“VehicleMgt Blueprint''}), \\ &(\text{“}ownership\text{''}, \text{“}string\text{''}, \text{“}AutoInc\text{''}), \ldots\} \\ e_2 := \{&\text{“ns:VehicleMgt-Off''}, \text{“}SaaSOffering\text{''}, \\ &(\text{“}offeringName\text{''}, \text{“}string\text{''}, \text{“Vehicle Management Software''}), \\ &(\text{“}offeringType\text{''}, \text{“}string\text{''}, \text{“}SaaS\text{''}), \ldots \\ &(\text{“}policy\text{''}, \text{“}PolicyProfile\text{''}, \text{“}ns : VehicleMgt - PP1\text{''}), \ldots\} \\ e_3 := \{&\text{“ns:VehicleMgt-Req1''}, \text{“}SaaSRequirement\text{''}, \\ &(\text{“}requirementName\text{''}, \text{“}string\text{''}, \text{“Context Information Service''}), \\ &(\text{“}requirementType\text{''}, \text{“}string\text{''}, \text{“}SaaS\text{''}), \ldots\} >> \\ e_4 := \{&\text{“ns:VehicleMgt-Req2''}, \text{“}PaaSRequirement\text{''}, \\ &(\text{“}requirementName\text{''}, \text{“SQL database''}), \\ &(\text{“}requirementType\text{''}, \text{“}string\text{''}, \text{“}PaaS\text{''}), \ldots\} >> \\ &\ldots \end{aligned} \tag{5.5}$$

Relations in M:

$$\begin{aligned} r_1 &:= (\text{“}hasOffering\text{''}, \text{“ns:VehicleMgt-BP''}, \text{“ns:VehicleMgt-Off''}) \\ r_2 &:= (\text{“}hasRequirement\text{''}, \text{“ns:VehicleMgt-BP''}, \text{“ns:VehicleMgt-Req1''}) \\ r_3 &:= (\text{“}hasRequirement\text{''}, \text{“ns:VehicleMgt-BP''}, \text{“ns:VehicleMgt-Req2''}) \\ r_4 &:= (\text{“}hasRequirement\text{''}, \text{“ns:VehicleMgt-BP''}, \text{“ns:VehicleMgt-Req3''}) \\ r_5 &:= (\text{“}horizontalLink\text{''}, \text{“ns:VehicleMgt-Off''}, \text{“ns:VehicleMgt-Req1''}) \\ r_6 &:= (\text{“}verticalLink\text{''}, \text{“ns:VehicleMgt-Off''}, \text{“ns:VehicleMgt-Req2''}) \\ &\ldots \end{aligned} \tag{5.6}$$

Finally, the Blueprint Model M:

$$M := \{\text{“http://www.eriss.org/VehicleMgt-BP\#''}, e_1, e_2, e_3, \ldots, r_1, r_2, r_3, \ldots\} \tag{5.7}$$

Please note that within the Blueprint Model M, the $r_{21}, r_{22}, r_{23}, r_{24}$ relations are the four examples of horizontal links that exist across two Blueprint Models. The target elements of these relations do not belong to M but the other Blueprint Models.

5.2.2 Formalizing a Blueprint Model as a RDF Graph

We choose to formalize a Blueprint Model as a *RDF graph* [Klyne & Caroll, 2004] due to the following two reasons:

- In the previous Section 4.4 we have formalized the BSL model as a set of OWL blueprint ontologies. By populating individuals of the OWL classes in these OWL blueprint ontologies, a blueprint model can be instantiated as another OWL model containing the instances of the OWL class. However, an OWL-based blueprint model is suitable only for the purpose of representing and reasoning on a blueprint.

  An OWL model is also a RDF Graph, since the OWL language extends the RDF vocabulary [Dean & Schreiber, 2004]. For the purpose of manipulating and exchanging data of an OWL model, its RDF Graph should be used [Patel-Schneider et al., 2004]. Hence, for the purpose of defining and formalizing the BMT operators to manipulate a blueprint model, we decide to use its RDF Graph representation.

- SPARQL [W3C, 2008] and SPARQL-Update [W3C, 2012] are the two well-established standards defining operations for manipulating and querying RDF Graphs. The use of RDF Graph as the formalization of a blueprint model will enable the formalization of the BMT operators using the SPARQL and SPARQL-Update operations.

- An RDF Graph is a labeled acyclic directed graph. The fact that the nodes of the graph can be labeled with the blueprint elements and their attributes, and the edges of the graph can be labelled with the relationship names (e.g. the horizontal and vertical links between the blueprint elements) provides an easier solution for the structural matching between two RDF graphs (i.e. two blueprint models). In fact, the structural matching between two RDF graphs will be used in the next Section 5.7 as the formalization of our *Match* operator introduced also in this section.

A RDF Graph is formally defined as a set of *RDF triples* [Klyne & Caroll, 2004]. A RDF triple contains three components: (1) subject as the URI reference of a *RDF resource*, (2) predicate as the URI reference of a *RDF property*, and (3) object as the URI of another RDF resource or a *RDF typed literal*.

Let us consider a RDF Graph G as the formalization of the blueprint model M with the following URI prefix declarations:

- bp:=<http://www.eriss.org/BlueprintOntology.owl#> : URI prefix declaration for the OWL Blueprint Ontologies

- rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#> : URI prefix declaration for the RDF syntax
- xsd:= <http://www.w3.org/2001/XMLSchema#> : URI prefix declaration for the XSD data types

Then, the mapping between a tuple-based representation of the blueprint model M and its RDF Graph G is explained in the following:

- The namespace $M.ns$ is formalized as the URI $G.uri$ of the RDF Graph G:

$$M.ns = G.uri$$

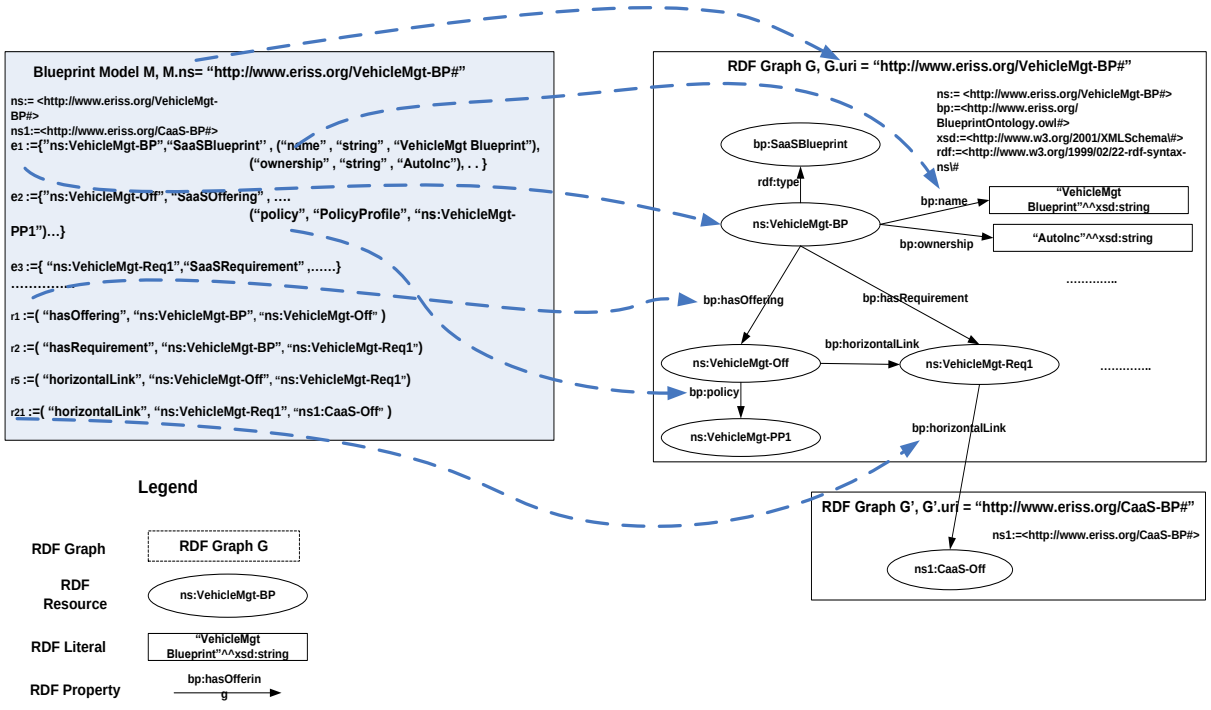
- An element $e = \{e.id, e.class, att^*\} \in M$ is formalized as the following RDF triple in G:

$$\text{“}e.id \quad rdf:type \quad bp:e.class\text{”}$$

- Each attribute $att = (att.name, att.type, att.value) \in e$ with a primitive type $att.type$ is formalized as the following triple in G:

$$\text{“}e.id \quad bp:att.name \quad att.value\text{^^xsd:}att.type\text{”}$$

Figure 5.4: Example of formalizing a Blueprint Model M as an RDF Graph G



- Each attribute $att = (att.name, att.type, att.value\} \in e$ with a complex type $att.type$ is formalized as the following triple in G:

$$\text{“}e.id \quad bp : att.name \quad att.value\text{”}$$

- Each relation $r = (r.name, r.id_s, r.id_t) \in M$ is formalized as the following triple in G:

$$\text{“}r.id_s \quad bp : r.name \quad r.id_t\text{”}$$

Figure 5.4 illustrates an example of mapping a tuple-based representation of a Blueprint Model M into a RDF Graph G. The Blueprint Model M specifying the `VehicleMgt-BP` blueprint can now be formally represented as a RDF Graph. The nodes in the graph G are either a RDF resource or a RDF typed literal, whilst an edge in G is a RDF property.

5.2.3 Implementation

The Jena Framework API is a Java API that supports the creation and manipulation of RDF Graphs[6]. In particular, its **com.hp.hpl.jena.rdf.model** package contains the following Java interfaces:

- The **Model** interface is used to implement a RDF Graph.
- The **Resource** interface is used to implement a RDF resource in a **Model**.
- The **Literal** interface is used to implement a RDF typed literal in a **Model**.
- The **Property** interface is used to implement a RDF property in a **Model**.
- The **Statement** interface is used to implement a RDF triple in a **Model**.

The concept of blueprint model has been developed as a class named **BlueprintModel** that implements the jena's **Model** interface. This class also contains a URI-typed namespace *ns* that serves as the unique ID of a blueprint model.

Furthermore, the Jena Framework also supports the serialization of a RDF graph into an XML document following the concrete RDF/XML syntax defined in [Beckett, 2004] and vice versa, a RDF Graph can be parsed from a RDF/XML document using the Jena Framework.

[6]Jena's Model Package: `http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/package-summary.html`

5.3 BMT Operators

The BMT operators have been developed based on the idea of model management operators supported in [Melnik, 2004]. In his thesis work [Melnik, 2004], Melnik has developed a set of generic model management operators that can assist users in manipulating any kinds of data model. In the previous Section 5.2 we have defined the concept of a blueprint model as a kind of data model specifying a cloud service. Hence, the idea of model management operators can easily be ported to the definition of the BMT operators that work on the blueprint models. In this section, we introduce an intuitive definition of the BMT operators. The selected BMT operators to be scrutinized in this thesis have been determined from the following two criteria:

- We have analyzed the work of Melnik in [Melnik, 2004], and select those operators that are significant for manipulating cloud service specifications.
- The concept of blueprint has been extensively used within the 4caaSt project [European Comission, 2010]. Within the project community, there is a desire of developing a toolset for the blueprint and the BMT operators clearly represent the desired functionalities that should be supported by this toolset. The selected BMT operators in this chapter reflect the desired functionalities of the blueprint toolset to support the CSBA developers in publishing, querying, and composing several blueprints available in the marketplace to fulfill their application requirements.

Table 5.1 introduces the list of the BMT operators together with the BMT techniques that they support:

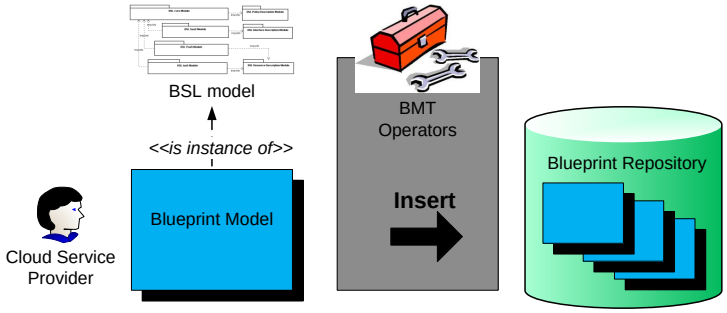
Table 5.1: BMT Operators supporting the BMT Techniques.

Supported Technique	**Operator's Signature**	**Operator's Functionality**
Publish a Blueprint to a Repository	***Insert*** *(BlueprintModel M)*	supports the insertion of a blueprint model *M* into a repository. The blueprint model *M* is identified in the repository with its namespace *M.ns* A cloud service provider, after specifying his blueprint in a blueprint model *M*, may want to use this operator to *publish* his blueprint model to a repository
Delete a Blueprint in a Repository	***Delete*** *(String ns)*	this operator supports the deletion of a blueprint model *M* in the repository based on its namespace *ns*. A cloud service provider may want to use this operator to *delete* his blueprint model in a repository by indicating the namespace of the blueprint model *M* to be deleted.

Query Blueprint(s) from a Repository	*BlueprintModel* **Query** *(String namespace)*	this operator returns a blueprint model from the repository that has the namespace as specified in the input parameter *namespace*. An CSBA Engineer may want to use this operator to *query* a blueprint model based on its namespace.
	*BlueprintModel** **Query** *(String offeringName)*	this operator returns all the blueprint models that contain an offering with the same name as the input parameter *offeringName*. An CSBA Engineer may want to use this operator to *query* the needed blueprint models based on their offering names.
	*BlueprintModel** **Query** *()*	this operator returns all the blueprint models in the repository. An CSBA Engineer may want to use this operator to *query* all the existing blueprint models.
Compose Blueprints	*Mapping** **Match** *(Requirement e, Offering e')*	this operator performs a matching between the requirement $e \in M$ and the offering $e' \in M'$ in two distinct blueprint model M and M'. The return contains (1) a set of mappings between an attributes of e and attributes of e', and (2) a mapping $e \mapsto e'$ if it exists. A mapping $e \mapsto e'$ indicates that the requirement $e \in M$ can be fulfilled by the offering $e' \in M'$. An CSBA Engineer may want to execute this operator to identify whether a mapping $e \mapsto e'$ exists between a requirement $e \in M$ and an offering $e' \in M'$.
	Link *(Requirement e, Offering e')*	This operator uses the *Match* operator to identify whether a mapping $e \mapsto e'$ exists. If this is the case, the operator then creates a "horizontal link" between the requirement $e \in M$ to the offering $e' \in M'$ to indicate that e is resolved by e'. An CSBA Engineer may want to use this operator to compose two blueprint models M and M' by resolving a requirement $e \in M$ with the offering $e' \in M'$ through a horizontal link.
	Unlink *(Requirement e, Offering e')*	this operator supports the deletion of a horizontal link between e and e', if there currently exists such a link.
	*BlueprintModel** **Resolve** *(BlueprintModel M)*	this operator supports an automatic composition of blueprint models to resolve all the requirements of an input blueprint model M. In particular, given the input blueprint model M, this operator: 1. uses the *Query* operator to retrieve all the current blueprint models from the repository. 2. composes M with the newly retrieved blueprint models using the *Link* operator, if possible. 3. The newly retrieved blueprint models may also contain requirements. This operator iteratively uses the *Link* operator to resolve all the requirements of these blueprint models. The return of this operator is a composition of blueprint models. If the return contains M, the *Resolve* operator is considered as "successful" and the return can be used as input for generating a deployment plan for M. If the return does not contain M, the *Resolve* operator is considered as "failed" and the return does not contain any useful information.

In the subsequent Sections, we will introduce each BMT operator in detail.

Figure 5.5: The *Insert* Operator



5.4 The *Insert* Operator

The conceptual definition of the *Link* operator is introduced in the subsection 5.4.1, followed by its formalization in subsection 5.4.2 and implementation in subsection 5.4.3.

5.4.1 Conceptual Definition

- Signature: ***Insert*** *(BlueprintModel M)*
- Input: the blueprint model M specifying a blueprint.
- Output: n/a.
- Fault: is returned if there is already another blueprint model M' in the repository that has the same namespace , i.e. $\exists M',\ M'.ns = M.ns$.
- Functionality: Figure 5.5 explains the functionality of the *Insert* operator. It supports a cloud service provider to insert a newly created blueprint model M into a repository. A cloud service provider, after using the BSL to specify his blueprint in a blueprint model, may want to use this operator to *publish* his blueprint model to a marketplace repository. This operator also makes sure that there exists no other blueprint model M' with the same namespace in the repository, otherwise a fault is returned.

5.4.2 Formalization

In this section, we formalize the semantics of the *Insert* operator. The input blueprint model M can be formalized as a RDF Graph G with the model namespace $M.ns$ as the graph ID $G.uri$, e.g. $G.uri = M.ns$. Therefore, the *Insert* operator can be formalized as the loading of an existing RDF Graph G into a RDF Graph Store using the SPARQL-Update operation "LOAD URI_from INTO GRAPH $G.uri$"

The SPARQL-Update statement "LOAD URI_from INTO GRAPH $G.uri$" supports the creation of a new empty graph with the ID $G.uri$ in the RDF Graph Store and then

reading the content of an existing RDF Graph G located at the URI_from URI into this new graph.. This LOAD statement returns a fault if there exists already a RDF graph with the same graph ID $G.uri$.

5.4.3 Implementation

Algorithm 5.1 Inserting a Blueprint Model

```
1: function INSERT(BlueprintModel M)
2:     URI_from: The URI of the RDF Graph of M.
3:     execute the following SPARQL-Update statement:
4:     "LOAD URI_from INTO GRAPH M.ns"
5:     if failure then
6:         return "Model already exists".
7:     end if
8: end function
```

The implementation of the *Insert* operator is explained in the pseudo algorithm 5.1 with the support of the Jena Framework for executing the SPARQL-Update statement. Input of the algorithm is a blueprint model M. In Line 3, the SPARQL-Update operation LOAD...INTO GRAPH is executed to create a new, empty RDF Graph in the target RDF Graph Store with the namespace of M as the new graph ID.

For implementing a RDF Graph Store, we create an experimental Graph Store using the Jena Fuseki server that can process SPARQL-Update statements.

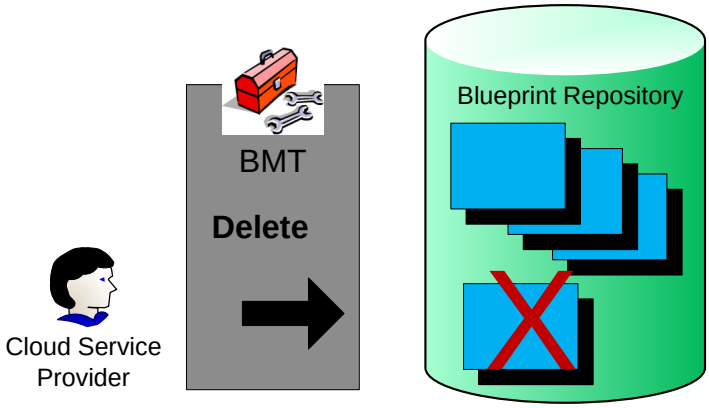
5.5 The *Delete* operator

The conceptual definition of the *Delete* operator is introduced in the subsection 5.5.1, followed by its formalization in subsection 5.5.2 and implementation in subsection 5.5.3.

5.5.1 Conceptual Definition

- Signature: ***Delete*** *(String namespace)*
- Input: the *namespace* of the blueprint model M that needs to be deleted.
- Output: n/a.
- Fault: is returned if there exists no blueprint model M with the input *namespace* , i.e. $\nexists M,\ M.ns = namespace$.

Figure 5.6: The Delete Operator



- Functionality: Figure 5.6 explains the functionality of the *Delete* operator. It supports the deletion of an existing blueprint model M in a repository, e.g. in case a deprecated blueprint model may need to be deleted upon the request of its provider. The deleted blueprint model M is identified by its namespace. Deleting the blueprint model M means deleting all of its elements $e^* \in M$ and relations $r^* \in M$.

5.5.2 Formalization

In this section, we formalize the semantics of the *Delete* operator. A blueprint model M can be formalized as a RDF Graph G. Therefore, the *Delete* operator can be formalized as the deletion of a RDF Graph G out of a RDF Graph Store using the SPARQL-Update operation "DROP GRAPH". In particular, the SPARQL-Update statement "DROP GRAPH *namespace*" supports the deletion of an existing RDF Graph G in a RDF graph store that has $G.uri = namespace$. This update statement returns a fault if there exists no RDF graph G with $G.uri = namespace$.

5.5.3 Implementation

Algorithm 5.2 Deleting a Blueprint Model

```
1: function DELETE(String namespace)
2:     execute the following SPARQL-Update statement:
3:     "DROP GRAPH namespace" # Deleting all the RDF Graph G with G.uri = namespace
4:     if failure then
5:         return "Model does not exist".
6:     else
7:         return "Model deleted successfully".
8:     end if
9: end function
```

The implementation of the *Delete* operator is explained in the pseudo algorithm 5.2. In Line 3, a SPARQL-Update statement is executed with the support of the Jena Framework API to delete a RDF Graph. We use the Jena Fuseki server as a RDF graph store that can process SPARQL-Update statements.

5.6 The *Query* Operator

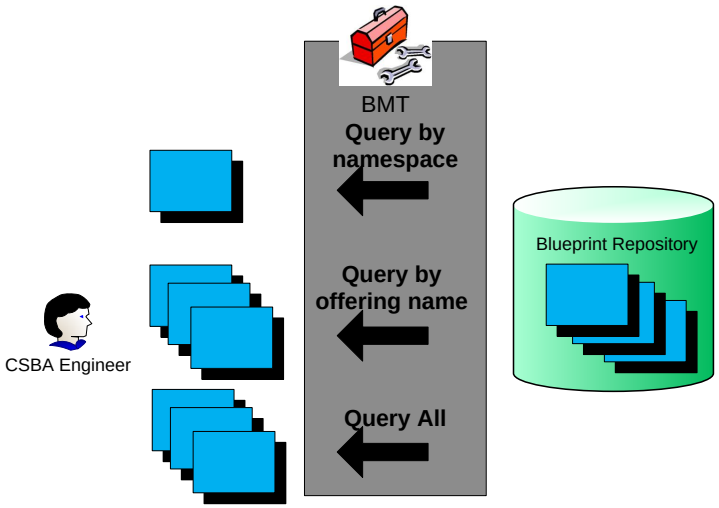
The conceptual definition of the *Query* operator is introduced in the subsection 5.6.1, followed by its formalization in subsection 5.6.2 and implementation in subsection 5.6.3.

5.6.1 Conceptual Definition

The *Query* operator has three variants that support different criteria for querying blueprint models.

- Signature:
  - Case 1: *BlueprintModel* | *null* ***Query****(String namespace)*
  - Case 2: *BlueprintModel** ***Query****(String offName)*
  - Case 3: *BlueprintModel** ***Query****()*
- Input: is either a namespace (Case 1) or an offering name (Case 2)
- Output: A set of blueprints that match the query criteria.
- Functionality: Figure 5.7 explains the functionality of the *Query* operator. The *Query* operator supports the query of *blueprint model(s)* from a repository that match the given criteria. We support the following query criteria:

Figure 5.7: The Query Operator and its Variants



- Querying based on the namespace: The namespace of a blueprint model is used as its unique ID in a repository. Hence, only one blueprint model that has a matched namespace may be selected and returned.
- Querying based on the offering name: Each blueprint model M contains only one offering

  $$e \in M,\ e.class = "SaaSOffering" | "PaaSOffering" | "IaaSOffering".$$

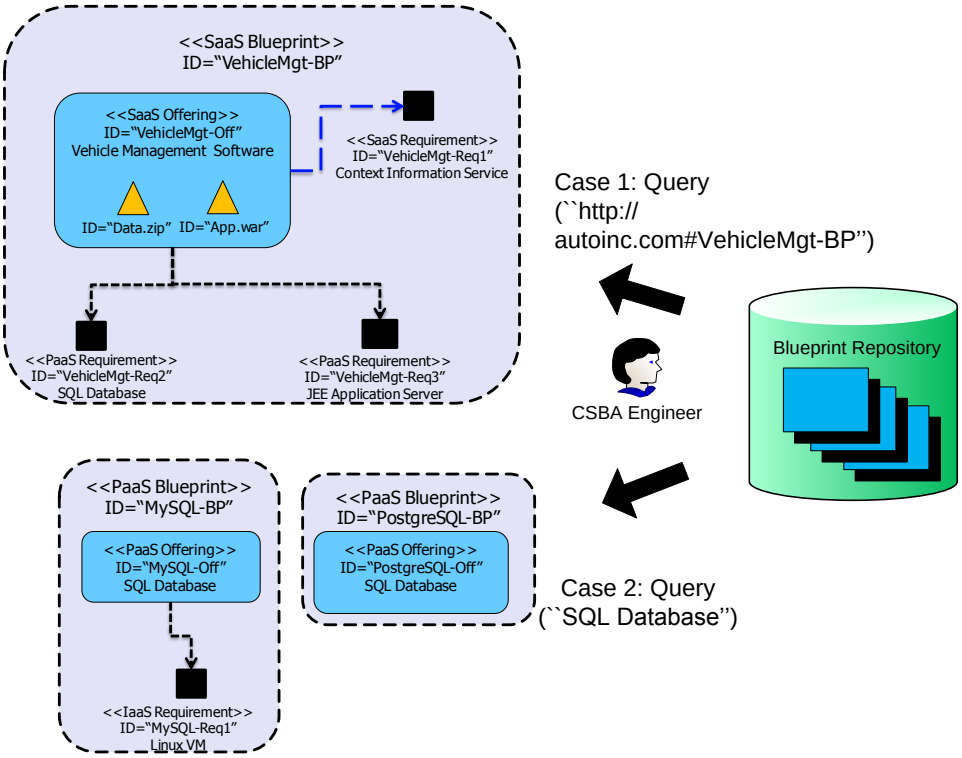
  The offering e is always defined with a mandatory attribute:

  $$att = << "offeringName", "string", att.value >>$$

  to specify the name of the offering. In this case, a blueprint model M is selected and returned if its $att.value$ can be matched with the input parameter *offName* by a simple string matching approach.
- Querying all current blueprint models in the repository.

Example 5.2 (Examples of Querying Blueprint Models) *Figure 5.8 illustrates two examples of using the* Query *operator in case 1 and case 2. In the first example, an CSBA engineer would like to query a blueprint model using its namespace. A namespace of a blueprint model serves as its globally unique ID. Hence, only one blueprint model "VehicleMgt-BP" is returned . In the second example, the CSBA engineer would like to query all the blueprint models that offers a "SQL Database". The return in this case is the two matched blueprint models "MySQL-BP" and "PostgreSQL-BP".*

Figure 5.8: Example of using the *Query* Operator



5.6.2 Formalization

In this section, we formalize the semantics of the *Query* operator. A blueprint model M can be formalized as a RDF Graph G. Therefore, all the three variants of the *Query* operator can be formalized as the query of RDF Graphs G_i from a RDF Graph Store using the SPARQL operations "SELECT" and "CONSTRUCT". In particular,:

- Case 1: The SPARQL query

  "CONSTRUCT { ?s ?p ?o } WHERE

  { GRAPH <*namespace*> { ?s ?p ?o }. }

  supports the query of an RDF Graph G that has its graph ID $G.uri = namespace$.

- Case 2: The SPARQL query

  "PREFIX bp:<http://www.eriss.org/BlueprintOntology.owl#>

  SELECT DISTINCT ?g WHERE

  { GRAPH ?g { ?s bp:hasOffering ?o . FILTER regex(?o, $offName$, "i").} }";

  supports the query of the graph IDs of all the RDF Graphs G_i that match the graph pattern "?s bp:hasOffering ?o . FILTER regex(?o, $offName$, "i").". This graph pattern contains a RDF triple "?s bp:hasOffering ?o", in which "?o" is a

RDF typed literal and can be matched with the input $offName$ using the $regex()$ string matching function supported by SPARQL.

Then, given the resulting list of graph IDs, the query operator in case 1 can be applied to subsequently query each RDF graph based on its ID.

- Case 3: The SPARQL query

  "SELECT DISTINCT ?g WHERE

  { GRAPH ?g { ?s ?p ?o .}";

  supports the query of all the graph IDs, since the graph pattern "?s ?p ?o" can be matched by any RDF Graph in the graph store.

  Then, given the resulting list of graph IDs, the query operator in case 1 can be applied to subsequently query each RDF graph based on its ID.

5.6.3 Implementation

Algorithm 5.3 Query a Blueprint Model based on its namespace

```
1: function QUERY(String namespace)
2:     # Create a new Blueprint Model M as the result of a SPARQL query
3:     BlueprintModel M = execute the following SPARQL query:
4:     "CONSTRUCT { ?s ?p ?o }
5:     WHERE
6:     { GRAPH <namespace> { ?s ?p ?o } . }"; # Get all the triples from the RDF Graph <namespace>
7:     M.ns = namespace; # Specify the namespace for M
8:     return M.
9: end function
```

The implementation of the *Query* operator in case 1 is explained in the pseudo algorithm 5.3. In Line 3, a blueprint model M can be created as the result of executing the SPARQL operation "CONSTRUCT". Then in Line 7, we specify the namespace for M.

Algorithm 5.4 Query Blueprint Models based on the name of its Offering

```
1:  function QUERY(String offName)
2:      List<BlueprintModel> result: list of the returned Blueprint Models;
3:      # Query all the namespaces of the matched Blueprint Models
4:      List<URI> allNSs = execute the following SPARQL query:
5:      " PREFIX bp:<http://www.eriss.org/BlueprintOntology.owl#>
6:      SELECT DISTINCT ?g WHERE
7:      { GRAPH ?g { ?s bp:hasOffering ?o . FILTER regex(?o, offName, "i").}}";
8:      for all (URI ns ∈ allNSs) do
9:          BlueprintModel M = Query(ns); # Query a Blueprint Model based on its namespace
10:         Add M to result;
11:     end for
12:     return result.
13: end function
```

The implementation of the *Query* operator in case 2 is explained in the pseudo algorithm 5.4. In Line 4, we execute the SPARQL operation "SELECT" to query the list of namespaces of the matched blueprint models. Then in Line 9, we use the implementation of the Query operator in case 1 (Algorithm 5.3) to query each Blueprint Model based on its namespace.

Algorithm 5.5 Query all Blueprint Models

```
1:  function QUERY
2:      List<BlueprintModel> result: list of the returned Blueprint Models;
3:      # Query the namespaces of all Blueprint Models
4:      List<URI> allNSs = execute the following SPARQL query:
5:      "SELECT DISTINCT ?g WHERE
6:      { GRAPH ?g { ?s ?p ?o .}}";
7:      for all (URI ns ∈ allNSs) do
8:          BlueprintModel M = Query(ns); # Query a Blueprint Model based on its namespace
9:          Add M to result;
10:     end for
11:     return result.
12: end function
```

The implementation of the *Query* operator in case 3 is explained in the pseudo algorithm 5.5. In Line 4, we execute the SPARQL operation "SELECT" to query the list of namespaces of all the Blueprint Models. Then in Line 8, we use the implementation of the Query operator in case 1 (Algorithm 5.3) to query each Blueprint Model based on its namespace.

5.7 The *Match* Operator

In this section, we discuss the behavior of applying the *Match* operator between a requirement e in a blueprint model M and an offering e' in another blueprint model M' with the constraint that e and e' need to be defined on the same layer of the cloud stack, i.e.:

$$\begin{aligned}
&(e.class = \text{“}SaaSRequirement\text{”} \ \wedge \ e'.class = \text{“}SaaSOffering\text{”}) \ \vee \\
&(e.class = \text{“}PaaSRequirement\text{”} \ \wedge \ e'.class = \text{“}PaaSOffering\text{”}) \ \vee \\
&(e.class = \text{“}IaaSRequirement\text{”} \ \wedge \ e'.class = \text{“}IaaSOffering\text{”}) \ \vee .
\end{aligned} \tag{5.8}$$

The goal of applying this operator is twofold:

1. to yield a set of *mappings* between the attributes $\{att^*\} \in e$ and the attributes $\{att'\}^* \in e'$. An mapping between an attribute $att \in e$ and an attribute $att' \in e'$, denoted as $att \mapsto att'$, has the meaning that att and att' have equivalent definitions (i.e. their names and types) and the value of att' is “better” than the value of att.

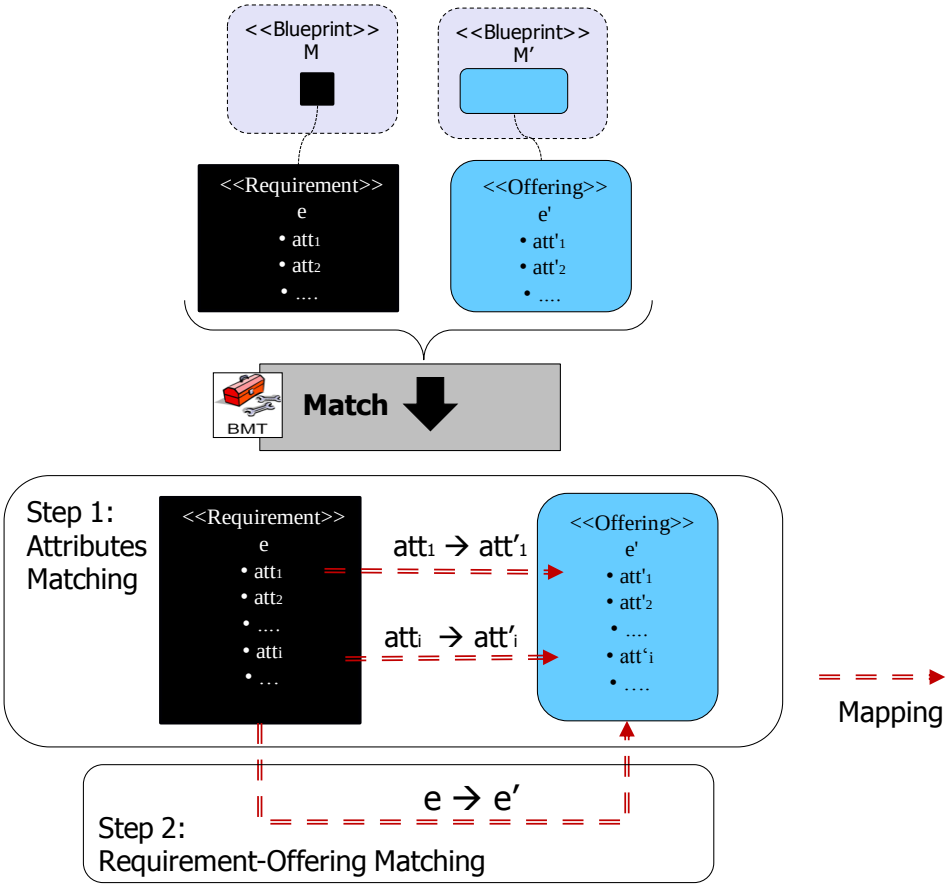
2. to identify whether a *mapping* from e to e', denoted as $e \mapsto e'$, exists. A mapping $e \mapsto e'$ has the meaning that the requirement $e \in M$ can be fulfilled by the offering $e' \in M'$.

The conceptual definition of the *Match* operator is introduced in the subsection 5.7.1, followed by its formalization in subsection 5.7.2 and implementation in subsection 5.7.3.

5.7.1 Conceptual Definition

- Signature: *Mapping** ***Match****(Requirement e, Offering e')*
- Input: a requirement e in the blueprint model M and an offering e' in another blueprint model M'. e and e' are defined on the same layer of the cloud stack.
- Output: is a set of mappings, which may contain:
  - a set of *attribute mappings* $(att \mapsto att')^*$, with $att \in e, att' \in e'$.
  - a *requirement-offering* mapping $e \mapsto e'$.
- Functionality: Figure 5.9 explains the functionality of the Match operator. Basically, it executes the following two steps:
  - Step 1 - Attributes Matching: The *Match* operator subsequently matches each pair of attributes (att, att') to identify whether an attribute mapping $att \mapsto att'$ exists.

Figure 5.9: The *Match* operator between a Requirement and an Offering



– Step 2- Requirement-Offering Matching : The *Match* operator uses the results of the attributes matching step to identify whether a requirement-offering mapping $e \mapsto e'$ exists.

In the following we explain the functionality of the *Match* operator in detail. The subsection 5.7.1.1 introduces the Step 1 - Attributes Matching. Then, the subsection 5.7.1.2 restricts the scope of Step 1 for brevity reason. Lastly, the Step 2 - Requirement-Offering Matching is explained in subsection 5.7.1.3.

5.7.1.1 Step 1: Attributes Matching

According to the BSL, a requirement e is defined with a set of attributes $\{att\}^*$, which contains (1) the core attributes defined by the BSL Core Module for specifying any kind of cloud service requirement, (2) the SaaS-specific attributes defined by the BSL SaaS Module for specifying a SaaS requirement, (3) the PaaS-specific attributes defined by the BSL PaaS Module for specifying a PaaS requirement, and (4) the IaaS-specific attributes defined by the BSL IaaS Module for specifying an IaaS requirement. Similarly,

the BSL also defines an equivalent set of core and layer-specific attributes $\{att'\}^*$ for specifying an offering e'. Table 5.2 lists the two sets of attributes $\{att\}^*$ and $\{att'\}^*$ for specifying a requirement e and an offering e' respectively. It also shows which attribute $att \in e$ can be matched with which attribute $att' \in e'$ and which matching technique can be used to derive a mapping $att \mapsto att'$. An important assumption for the attribute matching between two blueprints in this section is that the two blueprints share the same syntax and semantics definition of the attributes, e.g. by using the same ontology definition for the attributes. Without this assumption, attribute matching would become much more complicated.

Table 5.2: Attributes Matching between a Requirement $e \in M$ and an Offering $e' \in M'$

Attributes of a Requirement $e \in M$	Attributes of an Offering $e' \in M'$	Matching Technique
	Core Attributes	
$att(\text{“}requirementName\text{”}, \text{“}string\text{”}, att.value)$	$att'(\text{“}offeringName\text{”}, \text{“}string\text{”}, att'.value)$	$att \mapsto att' \iff att.value = att'.value$ (String Matching)
$att(\text{“}requirementType\text{”}, \text{“}string\text{”}, att.value)$	$att'(\text{“}offeringtype\text{”}, \text{“}string\text{”}, att'.value)$	$att \mapsto att' \iff att.value = att'.value$ (String Matching)
$att(\text{“}maxNrOfInstances\text{”}, \text{“}int\text{”}, att.value)$	$att'(\text{“}maxNrOfInstances\text{”}, \text{“}int\text{”}, att'.value)$	$att \mapsto att' \iff att.value \leq att'.value$ (Integer Comparison)
$att(\text{“}minNrOfInstances\text{”}, \text{“}int\text{”}, att.value)$	$att'(\text{“}minNrOfInstances\text{”}, \text{“}int\text{”}, att'.value)$	$att \mapsto att' \iff att.value \leq att'.value$ (Integer Comparison)
$att(\text{“}multitenant\text{”}, \text{“}boolean\text{”}, att.value)$	$att'(\text{“}multitenant\text{”}, \text{“}boolean\text{”}, att'.value)$	$att \mapsto att' \iff att.value = att'.value$ (Boolean Matching)
$att(\text{“}requiredPolicy\text{”}, \text{“}PolicyProfile\text{”}, att.value)$	$att'_j(\text{“}offeredPolicy\text{”}, \text{“}PolicyProfile\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}PolicyProfile\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Policy Profile Matching)
$att(\text{“}ext_property\text{”}, \text{“}ExtendedProperty\text{”}, att.value)$	$att'_j(\text{“}ext_property\text{”}, \text{“}ExtendedProperty\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}ExtendedProperty\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Extended Property Matching)
	SaaS-specific Attributes	
$att(\text{“}category\text{”}, \text{“}SaaSCategory\text{”}, att.value)$	$att'(\text{“}category\text{”}, \text{“}SaaSCategory\text{”}, att'.value)$	$att \mapsto att' \iff att.value = att'.value$ (String Matching)
$att(\text{“}requiredInterface\text{”}, \text{“}InterfaceDescrtipion\text{”}, att.value)$	$att'(\text{“}providedInterface\text{”}, \text{“}InterfaceDescription\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}InterfaceDescription\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Interface Description Matching)
	PaaS-specific Attributes	

$att(\text{“}product\text{”}, \text{“}PlatformProduct\text{”}, att.value)$	$att'(\text{“}product\text{”}, \text{“}PlatformProduct\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}PlatformProduct\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Platform Product Matching)
$att(\text{“}technology\text{”}, \text{“}PlatformTechnology\text{”}, att.value)$	$att'(\text{“}technology\text{”}, \text{“}PlatformTechnology\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}PlatformTechnology\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Platform Technology Matching)
$att(\text{“}resourceDescription\text{”}, \text{“}ResourceProfile\text{”}, att.value)$	$att'(\text{“‘}resourceDescription\text{”}, \text{“}ResourceProfile\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}ResourceProfile\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Resource Profile Matching)
IaaS-specific Attributes		
$att(\text{“}resourceDescription\text{”}, \text{“}ResourceProfile\text{”}, att.value)$	$att'(\text{“‘}resourceDescription\text{”}, \text{“}ResourceProfile\text{”}, att'.value)$	$att \mapsto att' \iff \exists \bar{e} \in M,\ \bar{e}.id = att.value,$ $\exists \bar{e}' \in M',\ \bar{e}'.id = att'.value,$ $\bar{e}.class = \bar{e}'.class = \text{“}ResourceProfile\text{”},$ $\bar{e} \mapsto \bar{e}'$ (Resource Profile Matching)
$att(\text{“}operatingSystem\text{”}, \text{“}string\text{”}, att.value) \in e$	$att'(\text{“}operatingSystem\text{”}, \text{“}string\text{”}, att'.value) \in e'$	$att \mapsto att' \iff att.value = att'.value$ (String Matching)
$att(\text{“}networkType\text{”}, \text{“}string\text{”}, att.value) \in e$	$att'(\text{“}networkType\text{”}, \text{“}string\text{”}, att'.value) \in e'$	$att \mapsto att' \iff att.value = att'.value$ (String Matching)

For a pair of attributes (att, att') that have primitive types “string”, “int”, and “boolean”, Table 5.2 introduces simple, straightforward matching techniques to identify a mapping $att \mapsto att'$, namely the string matching, the comparison for integer values, and the boolean matching.

For a pair of attributes (att, att') that are defined with a complex type, e.g. “PolicyProfile”, “ResourceProfile”, “InterfaceDescription”, etc., the attribute values $att.value$ and $att'.value$ are the URI reference to other elements in the Blueprint Models M and M'. Matching (att, att') in these cases leads to the matching between two other elements $\bar{e} \in M,\ \bar{e}.id = att.value$ and $\bar{e'} \in M',\ \bar{e'}.id = att'.value$, since a mapping $att \mapsto att'$ exists if and only if a mapping $\bar{e} \mapsto \bar{e'}$ exists. More sophisticated matching techniques are required to derive a mapping $att \mapsto att'$ in these cases. In the following, we briefly discuss the matching techniques that have been introduced in Table 5.2:

- *Policy Profile Matching*: The *Policy Profile* concept has been defined in the BSL Policy Description Module as a set of *Policy Assertions* that specify the value range for a certain *Policy Property*. Matching a Policy Profile $\bar{e} \in M$ with a Policy Profile $\bar{e}' \in M'$ aims to identify whether a mapping $\bar{e} \mapsto \bar{e}'$ exists. A mapping $\bar{e} \mapsto \bar{e}'$ means that any policy assertion specified in $\bar{e}$ can be fulfilled by at least a policy assertion specified in $\bar{e}'$.

In [Andrikopoulos et al., 2010], the authors have developed a matching technique between two QoS assertions by relatively positioning their intervals (value ranges) on their dimension of their common QoS property. Although this work focuses particularly on the QoS assertions, the technique can be reused to match two policy assertions in two distinct policy profiles.

- *Resource Profile Matching*: On the PaaS and IaaS layer, the *Resource Profile* concept has been defined in the BSL Resource Description Module as a set of *Resource Assertions* that specify the value range for a certain *Resource Property*. Matching a Resource Profile $\bar{e} \in M$ with another Resource Profile $\bar{e}' \in M'$ aims to identify whether a mapping $\bar{e} \mapsto \bar{e}'$ exists. A mapping $\bar{e} \mapsto \bar{e}'$ implies that a resource assertion in $\bar{e}$ can be fulfilled by at least a resource assertion in $\bar{e}'$.

  The Resource Profile Matching follows the same principle of Policy Profile Matching. Hence, the technique introduced in [Andrikopoulos et al., 2010] can also be reused.

- *Interface Description Matching*: On the SaaS layer, the *Interface Description* concept has been defined by the BSL Interface Description Module as a language construct used to specify both the *Signature* and the *Protocol* of a SaaS requirement (i.e. the required interface) or of a SaaS offering (i.e. the provided interface). Matching a required interface description $\bar{e} \in M$ with a provided interface description $\bar{e}' \in M'$ aims to verify the interface compatibility, i.e. whether the provided interface $\bar{e}'$ is compatible with the required interface $\bar{e}$. This matching comprises of the matching of their signatures and protocols.

  Since our BSL Interface Description Model has been developed based on the ASD Meta-model in [Andrikopoulos, 2010], the matching of two Interface Descriptions relies on the technique developed in [Andrikopoulos, 2010] that supports verifying the compatibility between a client's required service interface with a provider's offered service interface.

- *Platform Product Matching*: According to the BSL, a Platform Product is defined with the following attributes: a name (string), vendor (string), version (string), release date (Date), and info (URI). A matching between two platform products comprises of the mandatory matching their names and vendors. If their versions and release dates are specified, they need to be matched as well.

- *Platform Technology Matching*: According to the BSL, a Platform Technology has a similar structure like a Platform Product. A matching between two platform technologies comprises of the mandatory matching their names and vendors. If their versions and release dates are specified, they need to be matched as well.

- *Extended Property Matching*: According to the BSL, an Extended Property is defined as an extension point of the BSL that allows users to incorporate external languages or specification schemas. Hence, the matching between two extended properties requires user's decisions or external matching mechanisms.

5.7.1.2 Restricting the Scope of Attributes Matching

In the previous section, we have discussed in Table 5.2 several techniques that should be applied for matching a pair of attributes (att, att'), $att \in e$, $att' \in e'$ between a requirement e in a blueprint model M and an offering e' in another blueprint model M'. We have shown that due to the complex structure of a requirement and an offering defined by the BSL, a matching between them involves several sophisticated matching techniques, e.g. Policy Profile Matching, Resource Profile Matching, Interface Description Matching, etc.

Due to the limited scope of the thesis, we do not discuss all these matching techniques in detail. Interested readers are refer to the existing matching approaches that have been mentioned in the previous section 5.7.1.1. The aim of this section is to restrict the scope of attributes matching between a requirement and an offering. In particular, we would like to simplify the structure definition of a requirement and an offering, rather use their definitions in the BSL. Then, the attributes matching between a requirement and an offering will be defined based on their simplified structure definition.

Table 5.3: Attributes Matching between a simplified requirement $e \in M$ and a simplified offering $e' \in M'$

Attributes of a Requirement $e \in M$	Attributes of an Offering $e' \in M'$	Matching Technique
	Core Attributes (mandatory)	
$att_1(\text{“}requirementName\text{”}, \text{“}string\text{”}, att_1.value)$	$att'_1(\text{“}offeringName\text{”}, \text{“}string\text{”}, att'_1.value)$	$att_1 \mapsto att'_1 \Longleftrightarrow att_1.value = att'_1.value$ (String Matching)
$att_2(\text{“}requirementType\text{”}, \text{“}string\text{”}, att_2.value)$	$att'_2(\text{“}offeringtype\text{”}, \text{“}string\text{”}, att'_2.value)$	$att_2 \mapsto att'_2 \Longleftrightarrow att_2.value = att'_2.value$ (String Matching)
	Newly defined SaaS-specific Attributes (optional)	
$att_3(\text{“}maxResponseTime(s)\text{”}, \text{“}int\text{”}, att_3.value)$	$att'_3(\text{“}maxResponseTime(s)\text{”}, \text{“}int\text{”}, att'_3.value)$	$att_3 \mapsto att'_3 \Longleftrightarrow att_3.value \leq att'_3.value$ (Integer Comparison)
$att_4(\text{“}minAvailability(\%)\text{”}, \text{“}int\text{”}, att_4.value)$	$att'_4(\text{“}minAvailability(\%)\text{”}, \text{“}int\text{”}, att'_4.value)$	$att_4 \mapsto att'_4 \Longleftrightarrow att_4.value \geq att'_4.value$ (Integer Comparison)
	Newly defined PaaS/IaaS-specific Attributes (optional)	

$att_5(\text{“}minCPUSpeed(Ghz)\text{”}, \text{“}int\text{”}, att_5.value)$	$att'_5(\text{“}minCPUSpeed(Ghz)\text{”}, \text{“}int\text{”}, att'_5.value)$	$att_5 \mapsto att'_5 \iff att_5.value \geq att'_5.value$ (Integer Comparison)
$att_6(\text{“}minMemory(Gb)\text{”}, \text{“}int\text{”}, att_6.value)$	$att'_6(\text{“}minMemory(Gb)\text{”}, \text{“}int\text{”}, att'_6.value)$	$att_6 \mapsto att'_6 \iff att_6.value \geq att'_6.value$ (Integer Comparison)
$att_7(\text{“}minBandwidth(Gbit/s)\text{”}, \text{“}int\text{”}, att_7.value)$	$att'_7(\text{“}minBandwidth(Gbit/s)\text{”}, \text{“}int\text{”}, att'_7.value)$	$att_7 \mapsto att'_7 \iff att_7.value \geq att'_7.value$ (Integer Comparison)
$att_8(\text{“}minCapacity(TB)\text{”}, \text{“}int\text{”}, att_8.value)$	$att'_8(\text{“}minCapacity(TB)\text{”}, \text{“}int\text{”}, att'_8.value)$	$att_8 \mapsto att'_8 \iff att_8.value \geq att'_8.value$ (Integer Comparison)

Table 5.3 introduces our simplified structure definition of an offering and a requirement, which contains the following types of attribute:

- We keep using the two core attributes defined by the BSL for specifying the name and type of a requirement or an offering.
  - $att_1(\text{“}requirementName\text{”}, \text{“}string\text{”}, att_1.value)$ and $att'_1(\text{“}offeringName\text{”}, \text{“}string\text{”}, att'_1.value)$: These are the mandatory attributes that specify the name of a requirement and an offering. They are considered as the most important pair of attributes to be matched.
  - $att_2(\text{“}requirementType\text{”}, \text{“}string\text{”}, att_2.value)$ and $att'_2(\text{“}offeringType\text{”}, \text{“}string\text{”}, att'_2.value)$: These are the mandatory attributes that specify the cloud layer of a requirement and an offering, i.e. SaaS, PaaS or IaaS. Matching this pair of attributes is important because the matching between a requirement and an offering can only be performed on the same cloud layer.
- On the SaaS layer, we acknowledge the significance of a policy specification for a SaaS requirement and a SaaS offering. Currently, the BSL allows for this type of specification to be defined in a *Policy Profile*. However, as discussed in the previous section 5.7.1.1, matching two policy profiles is a complicated task that has been addressed by existing approaches. Hence, to simplify the matching between two policy specifications, we extend the structure of a SaaS requirement and a SaaS offering defined by the BSL with the following policy-related attributes:
  - $att_3(\text{“}maxResponseTime(s)\text{”}, \text{“}int\text{”}, att_3.value)$ and $att'_3(\text{“}maxResponseTime(s)\text{”}, \text{“}int\text{”}, att'_3.value)$: These two attributes specify the maximum response time (measured in seconds) of a SaaS requirement and a SaaS offering.

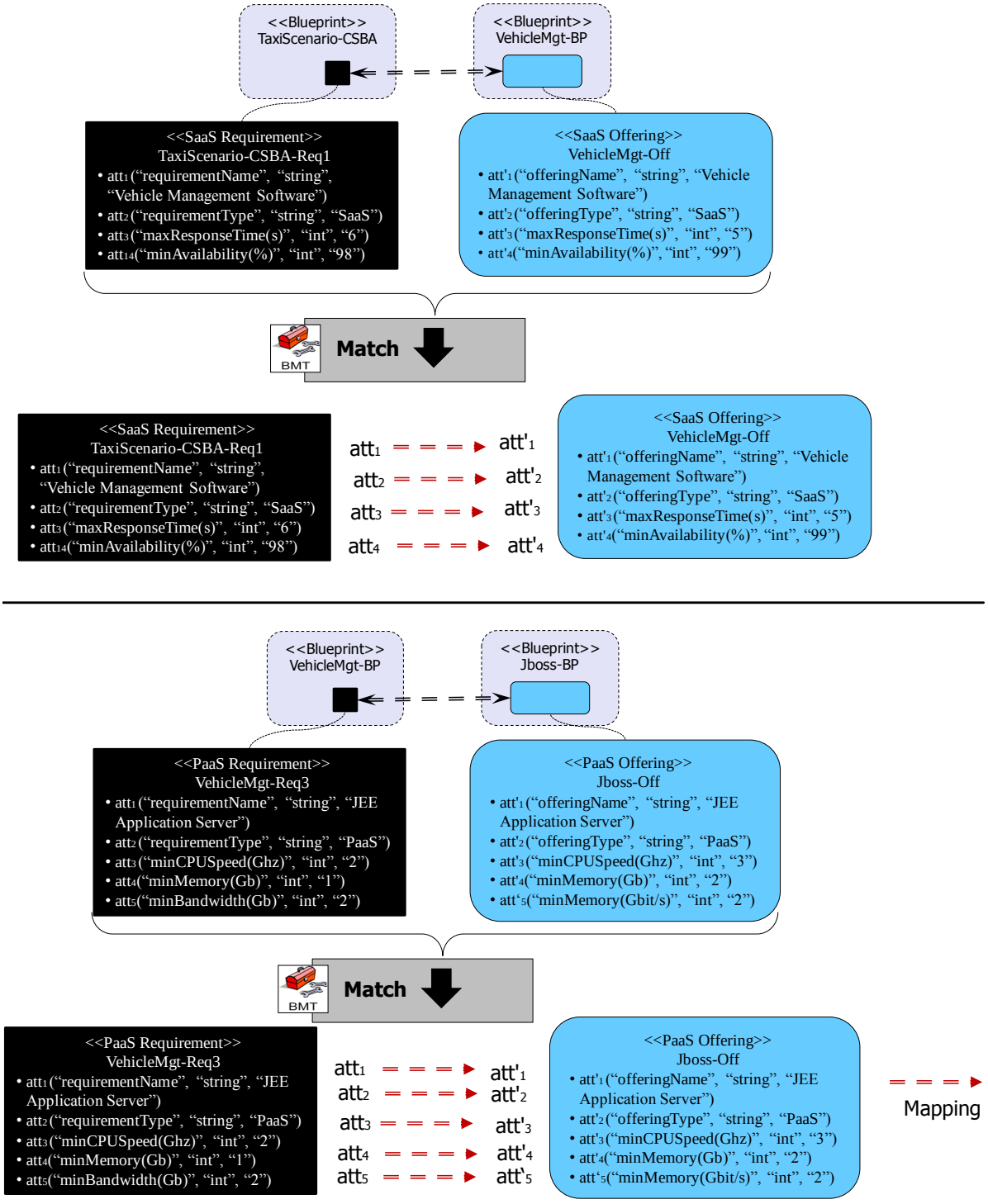
- $att_4(\text{“}minAvailability(\%)\text{”}, \text{“}int\text{”}, att_4.value)$ and $att'_4(\text{“}minAvailability(\%)\text{”}, \text{“}int\text{”}, att'_4.value)$: These two attributes specify the minimum availability (measured in %) of a SaaS requirement and a SaaS offering.

- On the PaaS/IaaS layers, we acknowledge the significance of a resource specification for a PaaS/IaaS requirement and a PaaS/IaaS offering. Currently, the BSL allows for this type of specification to be defined in a *Resource Profile*. However, as discussed in the previous section 5.7.1.1, matching two resource profiles is similar to the matching between two policy profiles and may require external matching techniques due to its complexity. Hence, to simplify the matching between two resource specifications, we extend the structure of a PaaS/IaaS requirement and a PaaS/IaaS offering defined by the BSL with the following resource-related attributes:

  - $att_5(\text{“}minCPUSpeed(Ghz)\text{”}, \text{“}int\text{”}, att_5.value)$ and $att'_5(\text{“}minCPUSpeed(Ghz)\text{”}, \text{“}int\text{”}, att'_5.value)$: These two attributes specify the minimum CPU Speed (measured in GHz) of a PaaS/IaaS requirement and a PaaS/IaaS offering. They are normally used for specifying a computing resource.
  - $att_6(\text{“}minMemory(Gb)\text{”}, \text{“}int\text{”}, att_6.value)$ and $att'_6(\text{“}minMemory(Gb)\text{”}, \text{“}int\text{”}, att'_6.value)$: These two attributes specify the minimum memory size (measured in Gbyte) of a PaaS/IaaS requirement and a PaaS/IaaS offering. They are normally used for specifying a computing resource.
  - $att_7(\text{“}minBandwidth(Gbit/s)\text{”}, \text{“}int\text{”}, att_7.value)$ and $att'_7(\text{“}minBandwidth(Gbit/s)\text{”}, \text{“}int\text{”}, att'_7.value)$: These two attributes specify the minimum network bandwidth (measured in Gbit/s) of a PaaS/IaaS requirement and a PaaS/IaaS offering. They are normally used for a network resource.
  - $att_8(\text{“}minCapacity(TB)\text{”}, \text{“}int\text{”}, att_8.value)$ and $att'_8(\text{“}minCapacity(TB)\text{”}, \text{“}int\text{”}, att'_8.value)$: These two attributes specify the minimum storage capacity (measured in Terabyte) of a PaaS/IaaS requirement and a PaaS/IaaS offering. They are normally used for a storage resource.

In summary, Table 5.3 has simplified the structure definition of a requirement e and an offering e' as the two sets of attributes $\{att_1, \ldots, att_8\}$ and $\{att'_1, \ldots, att'_8\}$ of primitive types “string” and “int”. Matching techniques for each pair (att_i, att'_i) have also been introduced in Table 5.3 to identify whether a mapping $att_i \mapsto att'_i$ exists. Through-

Figure 5.10: Examples of Attributes Matching between a simplified Requirement and a simplified Offering



out the rest of this chapter we will use this simplified version of matching between a requirement and an offering.

Example 5.3 (Attributes Matching between a simplified Requirement and a simplified Offering) *Figure 5.10 illustrates two examples of applying the Match operator to yield the attributes matching between a requirement and an offering. The left side of the Figure illustrates the matching between the SaaS Requirement "TaxiScenario-CSBA-Req1" and the SaaS offering "VehicleMgt-Off". This matching results into a set of attribute mappings indicating that all the attributes of the SaaS requirement can be mapped to the attributes of the SaaS offering. The right side of the Figure illustrates a different example of attributes matching between the PaaS requirement "VehicleMgt-Req3" and the PaaS offering "Jboss-Off".*

5.7.1.3 Step 2: Requirement-Offering Matching

This section introduces the second step of executing the *Match* operator between a requirement e and an offering e' to identify whether a mapping $e \mapsto e'$ exists. Based on the results of attributes matching between e and e' in Step 1, we define in the following the deduction rules to derive a mapping $e \mapsto e'$. *Please note that these rules are defined based on the simplified structure definition of a requirement and an offering in the previous section 5.7.1.2.*

A mapping $e \mapsto e'$ exists if both of the following rules are satisfied:

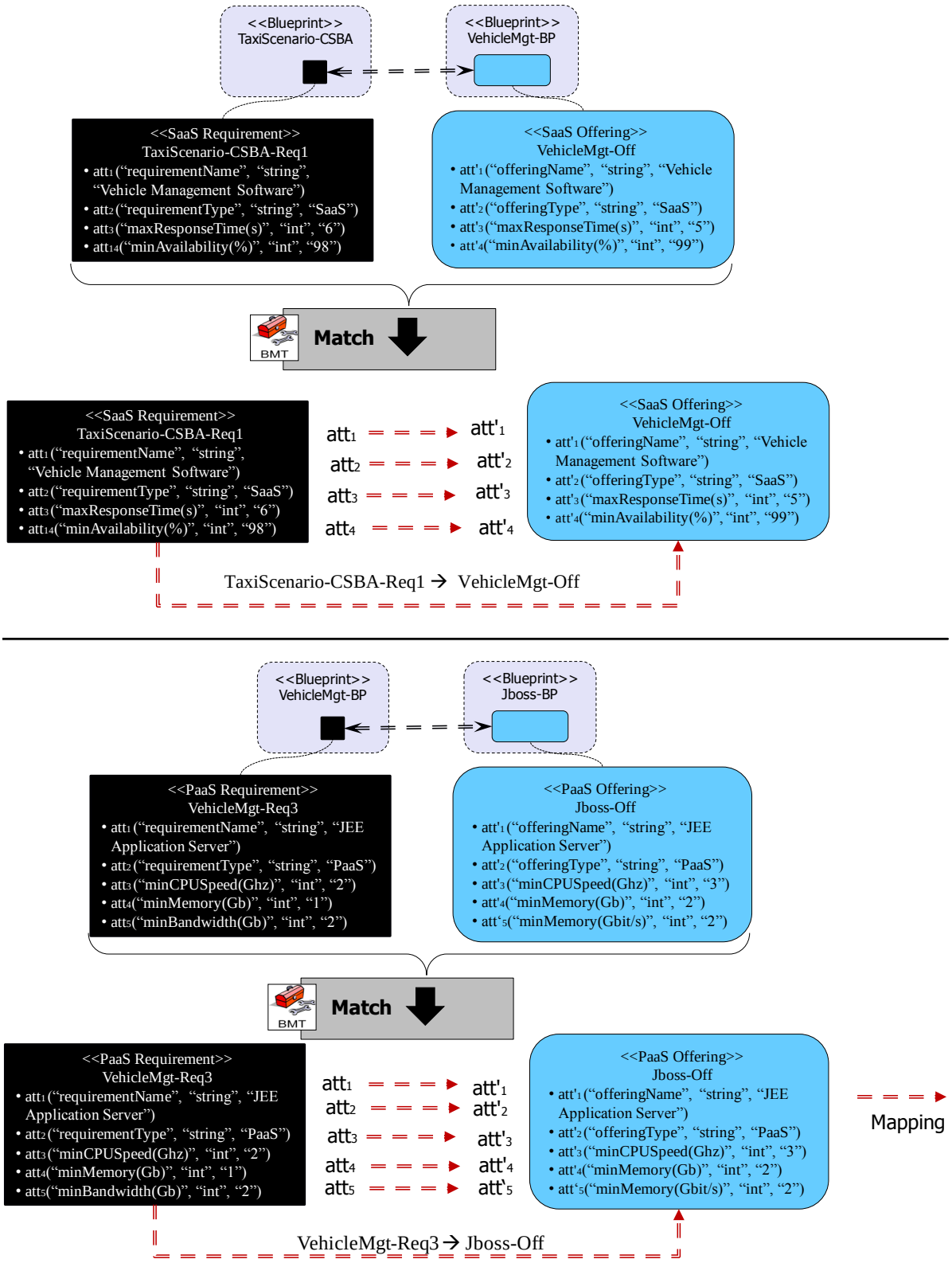
- $\exists att_1 \mapsto att'_1$ & $\exists att_2 \mapsto att'_2$: There exists mappings between their core attributes, i.e. $att_1.name = "requirementName"$ & $att'_1.name = "offeringName"$ and $att_2.name = "requirementType"$ & $att'_2.name = "offeringType"$.
- $\exists att_i \in e,\ 3 \leq i \leq 8 \implies (\exists att'_i \in e') \wedge (att_i \mapsto att'_i)$: If an optional attribute $att_i,\ 3 \leq i \leq 8$, is specified for the requirement e, then an equivalent attribute att'_i must also be specified for the offering e' and there must exist a mapping $att_i \mapsto att'_i$. For instance,

  $\exists att_3 \in e,\ att_3.name = "maxResposneTime" \implies$

  $(\exists att'_3 \in e,\ att'_3.name = "maxResposneTime")$ & $(att_3 \mapsto att'_3)$.

Example 5.4 (Matching between a Requirement and an Offering) *Figure 5.11 follows up the Example 5.3 to show that the SaaS Requirement "TaxiScenario-CSBA-Req1" can be mapped to the SaaS offering "VehicleMgt-Off", since there exist mappings not only between their core attributes, i.e. "requirementName" vs "offeringName" and "requirementType" vs "offeringType", but also between their optional attributes, e.g. "maxResponseTime" vs "maxResponseTime"and "minAvailability" vs "minAvailability". Similarly, the right side of the Figure also yields a mapping from the PaaS requirement "VehicleMgt-Req3" to the PaaS offering "Jboss-Off".*

Figure 5.11: Mapping Identification between a Requirement and an Offering



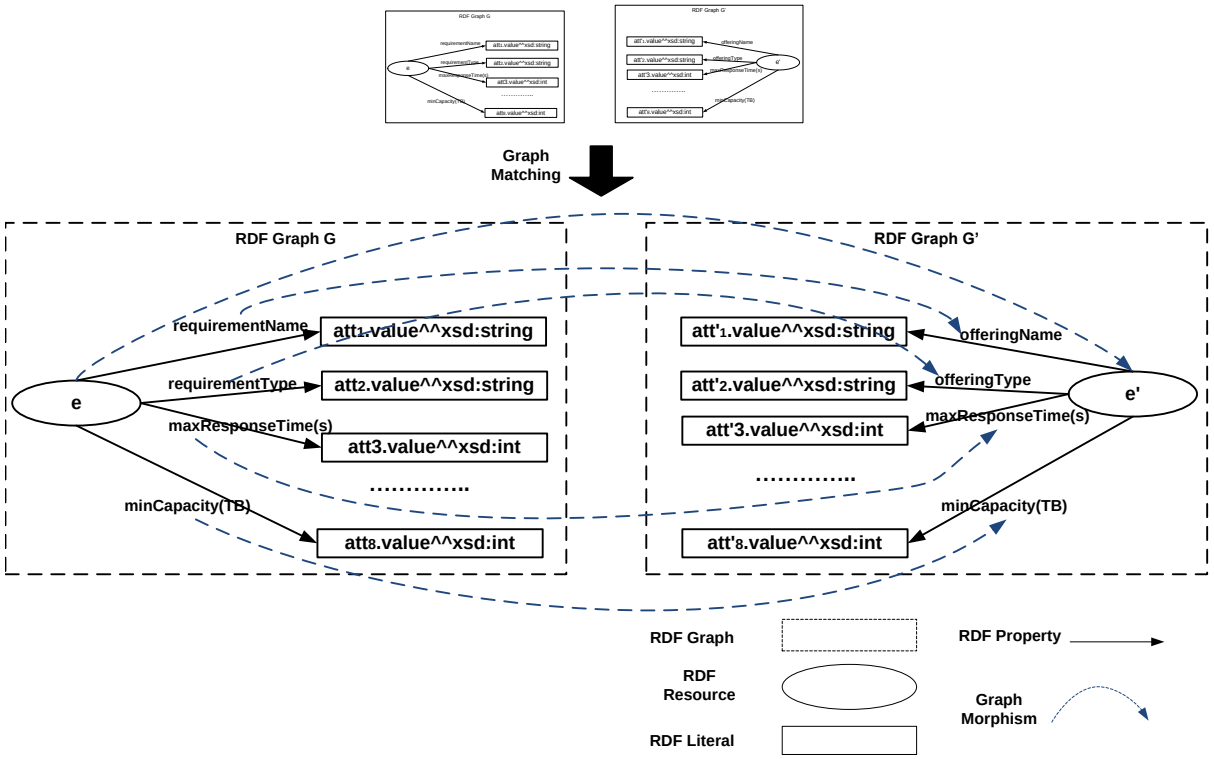
5.7.2 Formalization

In this section, we formalize the semantics of the *Match* operator when applying it between a requirement e in a blueprint model M and an offering e' in another blueprint model M'. The two blueprint models M and M' can be formalized as the two RDF Graphs G and G', in which e and e' can be formalized as a RDF resources (graph nodes) in G and G' respectively. Attributes of e and e' can be formalized as the RDF properties (adjacent edges of e and e') in G and G'. Figure 5.12 illustrates the formalization of the blueprint models M and M', the elements $e \in M$, $e' \in M'$, as well as all the attributes $att^* \in e$ and $att'^* \in e'$.

Figure 5.12 also illustrates that the *Match* operator can be formalized as a graph matching operator that works on the two nodes $e \in G$ and $e' \in G'$ of two distinct graphs G and G' and returns a set of graph morphisms, which may contain:

- A set of graph morphisms between the adjacent edges of e and the adjacent edges of e'.
- A graph morphism between e and e'.

Figure 5.12: Formalizing a Matching between a Requirement and an Offering as a set of Graph Morphisms



5.7.3 Implementation

Algorithm 5.6 Matching between a Requirement and an Offering using the *Match* operator

1: **function** MATCH(Requirement e, Offering e')
2: List<Mapping> $mappingList$: the list storing the mappings for the return;
3: # Step 1: Attributes Matching
4: **for all** $att_i \in e,\ att'_i \in e'$ **do**
5: Matching follows the rules defined in Table 5.3;
6: **if** $\exists(att_i \mapsto att'_i)$ **then**
7: Add $(att_i \mapsto att'_i)$ to $mappingList$;
8: **end if**
9: **end for**
10: # Step 2: Requirement-Offering Matching
11: **if** $(\exists(att_1, att'_1) \in mappingList)$ & $(\exists(att_2, att'_2) \in mappingList)$ **then** # 1st priority matching: the core attributes "requirementName" vs "offeringName', and "requirementType" vs. "offeringType"
12: **for all** $att_i \in e, 3 \leq i \leq 8$ **do** # For all *optional* attributes of e, e.g. "maxResposneTime", "minBandwidth", "minCPUSpeed",...
13: **if** $\nexists(att_i \mapsto att'_i) \in mappingList$ **then** # if an attribute mapping does not exist
14: **return** $mappingList$; # Then just return the mapping list containing only the attributes mappings
15: **end if**
16: **end for**
17: Add $e \mapsto e'$ to $mappingList$; # If all attribute mappings exists, then a mapping $e \mapsto e'$ also exists
18: **end if**
19: **return** $mappingList$; # Return the list of attribute mappings with or without $e \mapsto e'$
20: **end function**

The implementation of the *Match* operator is explained in the pseudo algorithm 5.6. From Line 4 to Line 9, the operator performs the *Step 1: Attributes matching* following the rules defined in Table 5.3. Each identified attribute mapping $att \mapsto att'$ is added into the list of mappings $mappingList$. Then in *Step 2: Requirement-Offering Matching* from Line 11 to Line 18, the operator evaluates the attributes matching results to identify whether a mapping between the requirement e and the offering e', i.e. $e \mapsto e'$, exists. The evaluation comprises of checking if there exist mappings between the core attributes of e and e' (Line 11), and also between their optional attribute (Line 12-15). Finally, if a mapping $e \mapsto e'$ exists, it will be added into the $mappingList$, which is the output of the operator.

By executing the *Match* operator, the user is able to identify all possible mappings between the attributes of e and the attributes of e', as well as the evaluation whether a

mapping $e \mapsto e'$ exists.

5.8 The *Link* and *Unlink* Operators

The conceptual definition of the *Link* and *Unlink* operators is introduced in the subsection 5.8.1, followed by their formalization in subsection 5.8.2 and implementation in subsection 5.8.3.

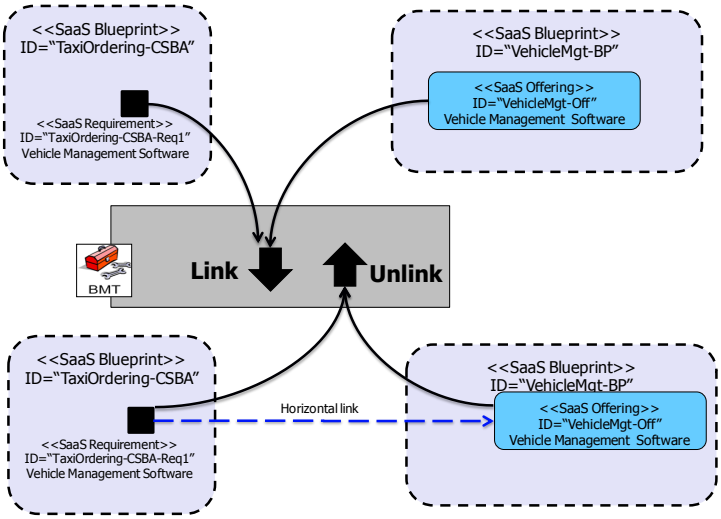
5.8.1 Conceptual Definition

- Signature:
  - ***Link*** *(Requirement e, Offering e')*
  - ***Unlink*** *(Requirement e, Offering e')*
- Input: a requirement e in a blueprint model M and an offering e' in another blueprint model M'.
- Output: n/a.
- Functionality: The *Link* operator supports the linking of a requirement $e \in M$ with an offering $e' \in M'$ through a horizontal link to indicate that e' is used to resolve e. Before creating a horizontal link, the *Link* operator uses the *Match* operator to check if e can be matched with e'. The *Unlink* operator is used for the opposite purpose. If there already exists a horizontal link between e and e', the *Unlink* simply removes it.

  The *Link* and *Unlink* operators can be used by a CSBA Engineer to compose and decompose two blueprint models through the horizontal links. Their definitions are simple. However, as an CSBA configuration has been defined in Definition 1.3, Section 1.2 as a composition of blueprint models, these two operators play an important roles in supporting the (re-)configuration of an CSBA.

Example 5.5 (Examples of using the Link and Unlink operators) *Figure 5.13 illustrates an example of using the* Link *and* Unlink *operators. First, the* Link *operator uses the* Match *operator to check if the SaaS requirement "TaxiOrdering-CSBA-Req1" can be matched with the SaaS offering "VehicleMgt-Off". According to our previous Example 5.4, this matching exists. Hence, the* Link *operator is able to create a horizontal link*

Figure 5.13: Examples of using the Link and Unlink Operators



between them.

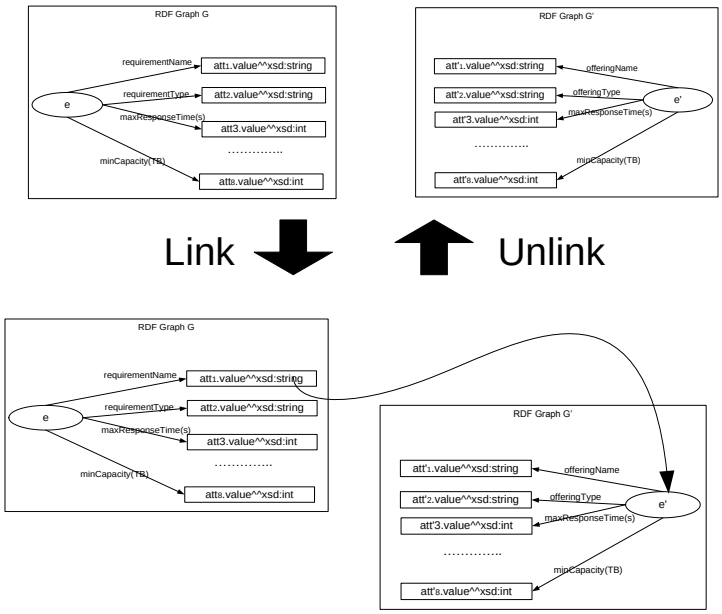
The effect of applying the Unlink *operator is opposite. It removes the horizontal link between the requirement "TaxiOrdering-CSBA-Req1" and the offering "VehicleMgt-Off".*

5.8.2 Formalization

In this section, we formalize the semantics of the *Link* and *Unlink* operators when applying them between a requirement e in a blueprint model M and an offering e' in another blueprint model M'. The two blueprint models M and M' can be formalized as the two RDF Graphs G and G', in which e and e' can be formalized as the two RDF resources (graph nodes) in G and G' respectively. Figure 5.14 illustrates the formalization of the blueprint bodels M and M' and the elements $e \in M$, $e' \in M'$.

Figure 5.14 also illustrates that the *Link* operator can be formalized as the creation of a cross-edge between the two graphs G and G'. The *Unlink* operator works in the other way around, i.e. it removes the cross-edge between G and G'.

Figure 5.14: Formalizing the Link and Unlink Operators



5.8.3 Implementation

Algorithm 5.7 Pseudo implementation of the *Link* and *Unlink* operators

```
1:  function LINK(Requirement e, Offering e')
2:      Let M be the BlueprintModel, e ∈ M;
3:      # Execute the Match operator between e and e' to get the list of mappings
4:      List<Mapping> mappingList = Match(e, e');
5:      # If there exists a mapping e ↦ e'
6:      if (e ↦ e') ∈ mappingList then
7:          # Create in M a "horizontalLink" relation between e and e'
8:          Create a new Relation r = ("horizontalLink", e.id, e'.id);
9:          Add r to M;
10:     end if
11: end function
12:
13: function UNLINK(Requirement e, Offering e')
14:     Let M be the BlueprintModel, e ∈ M;
15:     if ∃ r ∈ M, r = ("horizontalLink", e.id, e'.id) then
16:         Delete r out of M;
17:     end if
18: end function
```

The implementation of the *Link* and *Unlink* operators is explained in the pseudo algorithm 5.7. In Line 4, the *Match* operator is executed between the requirement e and the offering e' to identify the mappings between them. In Line 6, the operator checks whether a mapping $e \mapsto e'$ exists. If this is the case, a new horizontal link is created as a new relation $r \in M$.

The pseudo implementaion of the *Unlink* operator is introduced from Line 13 to Line 18. It simply identifies whether a horizontal link $r \in M$ already exists between the requirement e and the offering e'. If this is the case, the horizontal link will be deleted out of M.

5.9 The *Resolve* Operator

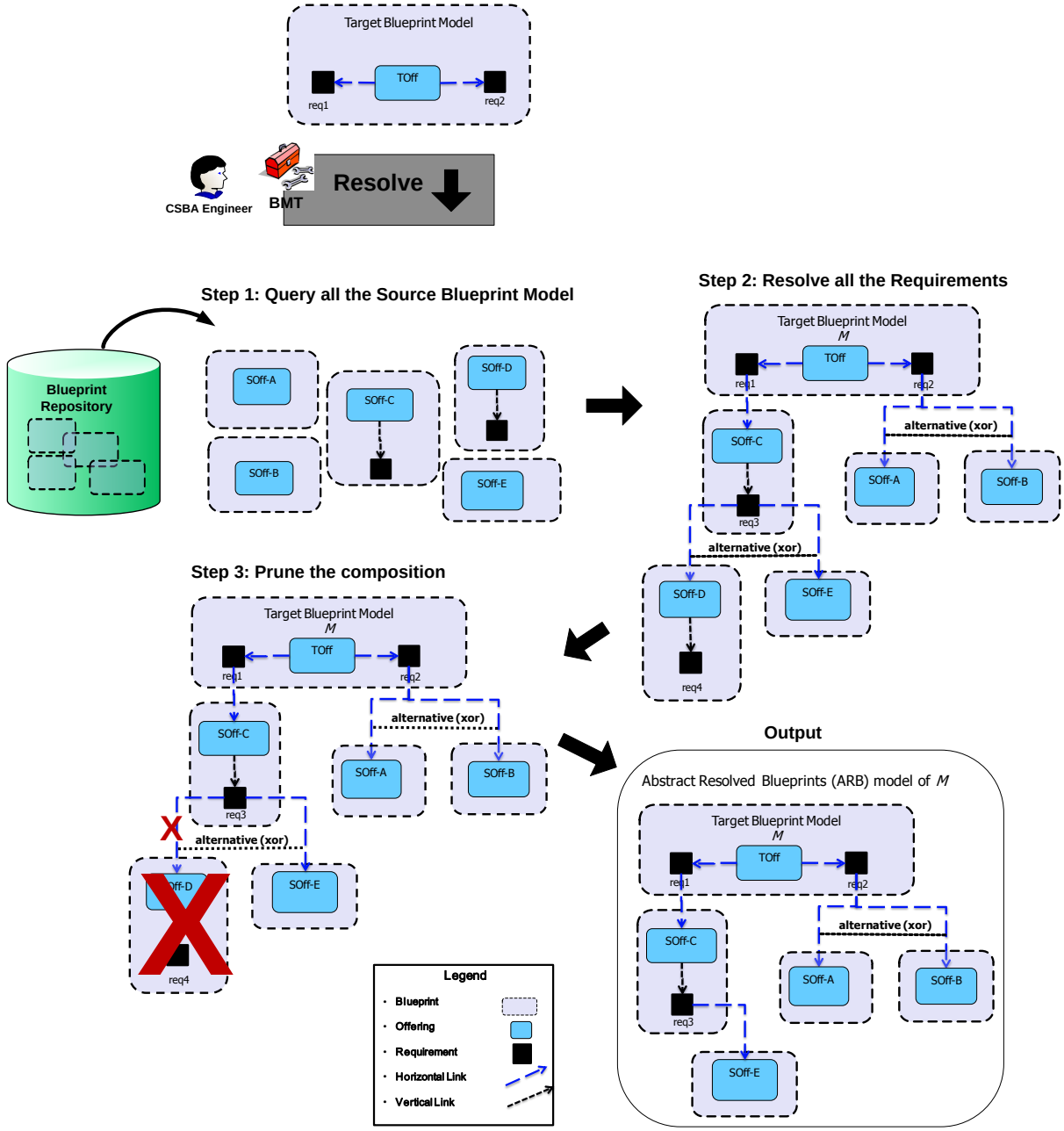
[7] This section introduces the *Resolve* operator for an automatic composition of Blueprint Models to form an end-to-end configuration of an CSBA. The conceptual definition of the *Resolve* operator is introduced in the subsection 5.9.1, followed by its formalization in subsection 5.9.2 and implementation in subsection 5.9.3.

5.9.1 Conceptual Definition

- Signature: *BlueprintModel** ***Resolve*** *(BlueprintModel M)*
- Input: The blueprint model M
- Output: A composition of blueprint models. As we have introduced in the previous Section 5.8, two blueprint models can be composed by the *Link* operator through a horizontal link.
- Functionality: Figure 5.15 explains the functionality of the *Resolve* operator. The input blueprint model M is called the *target blueprint model*. The *Resolve* operator is used to resolve the target blueprint model, which means to resolve all of its requirements, i.e. `req1` and `req2`. Resolving a requirement means to find an offering that can fulfill the requirement and use the *Link* operator to create a horizontal link between them. The *Resolve* operator tries to resolve the target bluperint model in the following three steps:
  - Step 1 - Query all the source blueprint models: this step queries all the blueprint models from the repository (called the *source blueprint models*), using the *Query* operator.

[7] The result presented in this section has been published in [Nguyen et al., 2012b]

Figure 5.15: The *Resolve* Operator



– Step 2 - Resolve all the requirements: this step tries to resolve the requirements of both the target blueprint model *M* and the source blueprint models. It uses the *Link* operator to create a horizontal link (if possible) between a requirement and an offering of two distinct blueprint models. After performing this step, a requirement may be resolved by only one offering, e.g, the requirement `req1` in Figure 5.15, or it may be resolved by two or more alternative offerings, e.g. the requirements `req2` and `req3` in Figure 5.15, or it may not be resolved at all, e.g. the requirement `req4` in Figure 5.15. The

result of this step is a composition of blueprint models that has been created by cross-linking the requirements with the offerings through the horizontal links.

– Step 3 - Prune the composition: By the end of step 2, a requirement may already be resolved by the *Link* operator, or it may still remain unresolved. This step subsequently deletes all the blueprint models that still contain unresolved requirements. For instance in Figure 5.15, the blueprint model containing the offering `Off-D` and the requirement `req4` is deleted since the requirement `req4` is still unresolved. Lastly, this step returns a composition of blueprint models that no longer contains any unresolved requirements.

If the returned composition of blueprint models does not contain the target blueprint model M, the resolution is considered as "failed" and the composition does not contain any useful information. In opposite, if the returned composition contains the target blueprint model M, the resolution is considered as "successful" and the returned composition of blueprint models is called the Abstract Resolved Blueprints (ARB) model of M. An ARB model of the target blueprint model M specifies all the possible composition of source blueprint models to resolve the requirements of M.

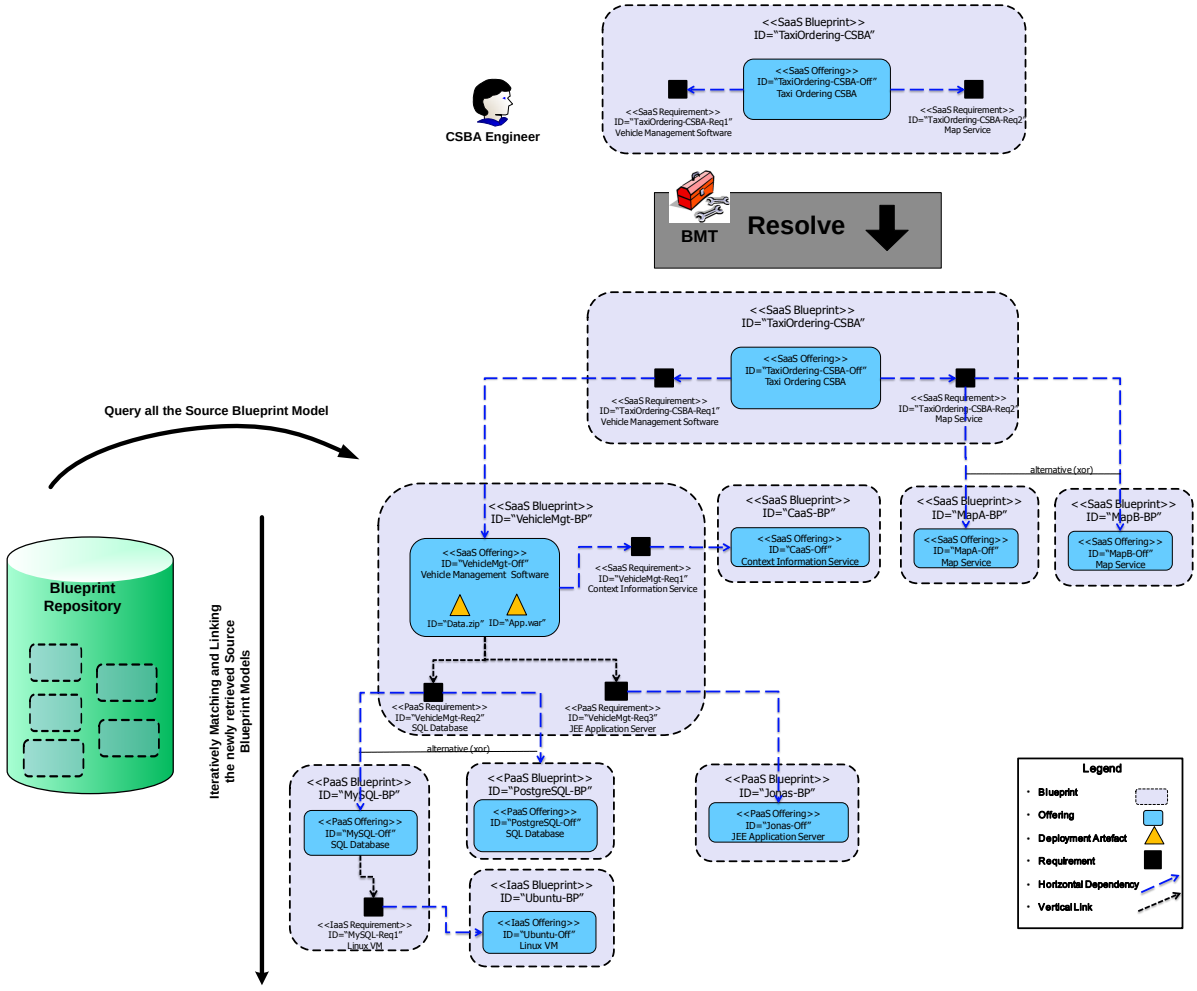
From the engineering point of view, an CSBA engineer can specify his CSBA in a target blueprint model M and then use the *Resolve* operator to generate an ARB model of M (if there exists any). The ARB model of M can then be used as the configuration of the CSBA's deployment environment using cloud services. Typically, it is used to generate a deployment plan for the CSBA. As an example, the TOSCA standard specification [OASIS, 2013] can be used at this point to specify the deployment topology and deployment plan for the CSBA.

Example 5.6 (Examples of using the *Resolve* operator) *Figure 5.16 illustrates an example of using the* Resolve *operator for the* `TaxiScenario-CSBA` *blueprint model. In Step 1, the* Resolve *operator uses the* Query *operator to query all the existing blueprint models in the repository, namely all the blueprint models introduced in our* Taxi Tilburg Scenario.

Then, in Step 2, the Resolve *operator uses the* Link *operator to create horizontal links between a requirement and an offering whenever a matching is found between them. As illustrated in Figure 5.16, the requirements* `TaxiOrdering-CSBA-Req2` *and* `VehicleMgt-Req2` *have two alternative horizontal links. The result of this step is a composition of blueprint models through the horizontal links.*

Since the result of step 2 does not contain any unresolved requirements, Step 3 does not need to perform

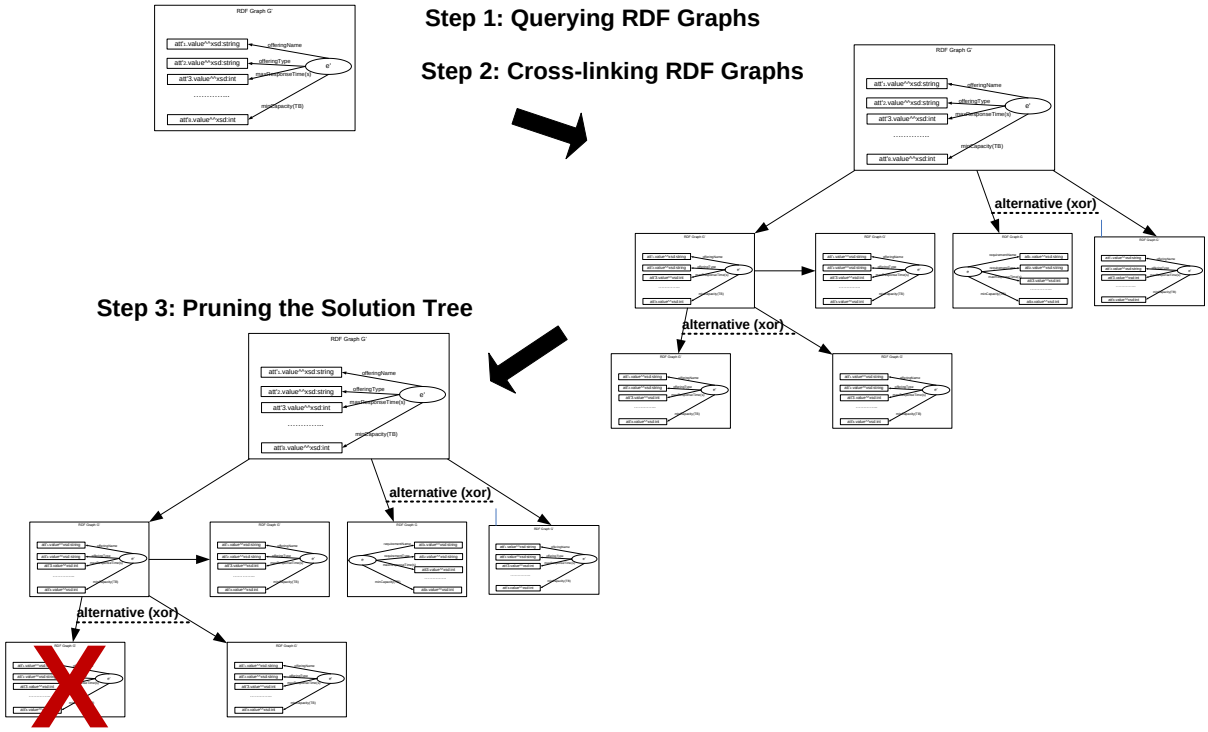
Figure 5.16: Example of applying the *Resolve* Operator



any pruning task. The result of step 2 is returned as an ARB model of the `TaxiScenario-CSBA` *blueprint model.*

Using the ARB model as the output of the Resolve *operator, the CSBA engineer of the* Taxi Tilburg *company is able to configure the deployment environment of the* TaxiScenario-CSBA. *More specifically, he knows which SaaSs, PaaSs, and IaaSs of external providers are needed by the* TaxiScenario-CSBA, *and in which order these cloud services should be deployed. For instance, the* `Jonas-BP` *blueprint needs to be deployed so that it can be used by the* `VehicleMgt-BP` *blueprint to deploy its software binaries. Either the* `MySQL-BP` *blueprint or the* `PostgreSQL-BP` *blueprint should be deployed so that the data of the* `VehicleMgt-BP` *can be managed. In case the* `MySQL-BP` *blueprint is selected, then the* `Ubuntu-BP` *blueprint needs to be deployed first, which provides a linux virtual machine hosting the* `MySQL-BP` *blueprint.*

Figure 5.17: Formalizing the *Resolve* Operator



5.9.2 Formalization

In this section, we formalize the semantics of the *Resolve* operator. The target blueprint model M can be formalized as a RDF graph G whilst all the source blueprint models M'_i can be formalized as the RDF Graphs G'_i. In the following, we explain the semantics of each step of the *Resolve* operator in terms of graph operations

- Step 1: The semantics of this Step has been formalized in Section 5.6 as the use of SPARQL operation to query all the RDF graphs G'_i from a RDF Graph Store.
- Step 2: The semantics of this Step has been formalized in Section 5.8 as the iterative cross-linking of two nodes in two distinct RDF graphs. The result of this step is an aggregated graph AG, in which G and all the G'_i are the subgraphs. Figure 5.17 illustrates the aggregated graph AG as the result of performing both Step 1 and Step 2.
- Step 3: The *Resolve* operator traverses through the aggregated graph AG, identifies a node of type "Requirement" that still has no "horizontalLink" edge. If such as node is identified, the subgraph containing it will be pruned. Finally, the aggregated graph AG is returned.

5.9.3 Implementation

The implementation of the *Resolve* operator is explained in the pseudo algorithm 5.8. In Line 5, source blueprint models are queried from the repository using the *Query* operator. These source blueprint models are stored in a list of blueprint models $listBMs$. In Line 6, the target blueprint model is added to $listBMs$. Line 5 and 6 are basically the implementation of Step 1 of the *Resolve* operator.

From Line 9 to Line 13, Step 2 of the operator is implemented. The operator traverses through all the requirements and offerings of all the blueprint models in the $listBMs$. For each pair of a requirement e and offering e', the *Link* operator is applied. A horizontal link may be created if e can be fulfilled by e'.

Step 3 of the operator is implemented from Line 17 to Line 19. The operator traverses again through all the requirements in all blueprint models in the $listBMs$ and identifies those that do not have a "horizontalLink" relation. If such a requirement is identified, the blueprint model containing it is deleted out of $listBMs$ using the helper function $DeleteBlueprintModel$.

Finally in Line 20, the $listBMs$ is returned as the output ARB.

The helper function $DeleteBlueprintModel$ is implemented from Line 25 to Line 31. It supports the deletion of a blueprint model M out of the list $listBMs$. Firstly in Line 26, it deletes all the "horizontalLink" relations that exists between other blueprint models and the offering of M. Then from Line 29 to Line 31, all elements and relations in M are deleted and finally M is deleted out of the $listBMs$.

Algorithm 5.8 Pseudo implementation of the *Resolve* operator

```
1: function RESOLVE(BlueprintModel targetBM)
2:     List<BlueprintModel> listBMs; # The list of Blueprint Models listBMs that will be returned as the resolution result
3:
4:     # Step 1: listBMs is initialized by querying all the source blueprint models from the repository
5:     listBMs = Query();
6:     Add targetBM to the listBMs;
7:
8:     # Step 2: Fulfill all the requirements of all Blueprint Models in listBMs
9:     for all Element e ∈ BlueprintModel M ∈ listBMs, e.type = "Requirement" do
10:        for all Element e' ∈ BlueprintModel M' ∈ listBMs, e'.type = "Offering" do
11:            Link(e, e'); # a horizontal link may or may not be created between the Requirement e and the Offering e'
12:        end for
13:    end for
14:
15:    # Step 3: Pruning the solution
16:    # Running a loop to detect a Blueprint Model M ∈ listBMs that still contains "unresolved" requirements
17:    while (∃ Element e ∈ BlueprintModel M ∈ listBMs, e.type = "Requirement") &
       (∄ Relation r ∈ M, r.name = "horizotnalLink", r.id_s = e.id) do
18:        DeleteBlueprintModel (M, listBMs); # Delete the BlueprintModel M using the help function
19:    end while
20:    return listBMs; # Return the list of Blueprint Models as the resolution result
21: end function
22:
23: # Help function to delete a blueprint model M out of the solution list listBMs
24: function DELETEBLUEPRINTMODEL(BlueprintModel M, List<BlueprintModel> listBMs)
25:    # Remove all the horizontal links from another Blueprint Model M' to the offering of M, i.e.:
26:    for all (Relation r' ∈ BlueprintModel M' ∈ listBMs, r'.name = "horizontalLink"
       and Element e ∈ M, e.type = "Offering" with r'.id_t = e.id) do
27:        Delete r
28:    end for
29:    Delete all Elements e* ∈ M;
30:    Delete all Relations r* ∈ M;
31:    Delete M out of listBMs;
32: end function
```

5.10 Discussion

This chapter has introduced a set of operators that collectively implement the proposed Blueprint Manipulation Techniques (BMTs). The original idea of developing this set of operators stems from the model management operators developed by Melnik [Melnik, 2004]. It is important to remark that this work aims only to propose a set of most essential operators that has been selected by the 4CaaSt consortium [European Comission, 2010]. The selected operators in this chapter reflect the desired functionalities of the blueprint toolset to support the CSBA developers in publishing, querying, and composing several blueprints available in the marketplace to fulfill their application requirements. In the future work we will continuously gather feedbacks from the cloud development community to extend this set of operators to a more complete toolset for cloud application development with the blueprints.

Given the current set of operators there have been already remarks and improvement suggestions among the 4CaaSt community, which we would like to discuss in the following:

- *More expressive query language*: This chapter has introduced the simplistic versions of the *Query* operator with the namespace and the offering name as input. The idea is only to demonstrate some simple definitions of the *Query* opertor. In the future work, a more expressive query language is expected to support more complex query options. In fact, currently with SPARQL the users can already form complex query expressions to query the blueprints from a repository. However, SPARQL is a quite technical language that is not familiar with people outside the semantic web community. A more user-friendly blueprint query language is expected at this point.

- *Operator for checking the consistency of the blueprints*: An instantiated blueprint may be modified by the user in different cases: e.g. updating its content, extending it with external languages, etc. These activities could lead to an inconsistency of a blueprint against the BSL syntax. Hence, an operator is needed to check the consistency coherence of blueprints to the BSL syntax.

- *A common ontology of attributes for matching*: Currently, the *Match* operator is performed on a common set of attributes defined by the BSL syntax. Given the situation that two blueprints with extensions need to be matched, then the *Match* operator could not function properly on the two extension parts, since they may be defined with different attribute structure. To prevent this problem, a common ontology of attributes could be developed in the future to support a more precise matching between blueprints. Then, the implementation of the *Match* operator will restrict the matching so that it only support the matching of attributes that exist in this ontology.

CHAPTER 6

VALIDATION

One of the most significant task in design science [Hevner et al., 2004] is the validation of the result to ensure its applicability in the real world. As we have discussed in the Introduction Chapter 1, section 1.5, the validation of the *Blueprint Approach* has been performed on four levels:

1. The **technical soundness** of the BSL is guaranteed by its mapping to a formal knowledge representation model described in RDF/OWL [McGuinness & van Harmelen (Eds.), 2004]- a well-established standard for describing Internet resources and semantic web. Based on this formal knowledge model, the operators defined by the BMT's have been formalized using SPARQL [W3C, 2008] and SPARQL-Update [W3C, 2012] operations, which are standards for manipulating RDF/OWL knowledge model. Using well-defined and widely-accepted standards for knowledge representation and manipulation ensures the logical consistency in our proposed Blueprint Approach.

2. The **usability** of the BSL and BMT's is demonstrated by using a running example throughout the previous chapters of the thesis. This example is borrowed from a real-case scenario that has been developed by the 4CaaSt community as one of the three validation scenarios in the 4CaaSt project [European Comission, 2010]. Reusing a scenario from the 4caaSt community shows that our solution solves a real-case defined by a group of cloud computing practitioners.

3. The **technical feasibility** of the BSL and BMT's is proved by an internal "proof-of-concept" prototype that has been implemented using well-known standardized languages and widely adopted tools.

4. The **practical validity** of our approach is exhibited through our participation in various industry prototypes that have been developed based on the proposed BSL and BMTs in the 4caaSt project. Within the scope of this project, the proposed Blueprint Approach has been adopted as one of its core contributions [Gómez et al., 2012]. Future cloud products of our industry partners in the project are also envisioned in this direction. Applying the Blueprint Approach in the 4caaSt prototypes helps validate the applicability of the Blueprint Approach in the cloud computing domain, since it can be proved to meet the expectations of our industry partners in the project.

Since the first two types of validation have been introduced in the previous chapters of the thesis, in this chapter we focus on the last two validation activities. In particular, Section 6.1 introduces a proof-of-concept prototype for the Blueprint Approach. Then in Section 6.2, we will report our activities in the 4caast project to demonstrate the **practical validity** of the Blueprint Approach.

6.1 Technical Feasibility of the Blueprint Approach

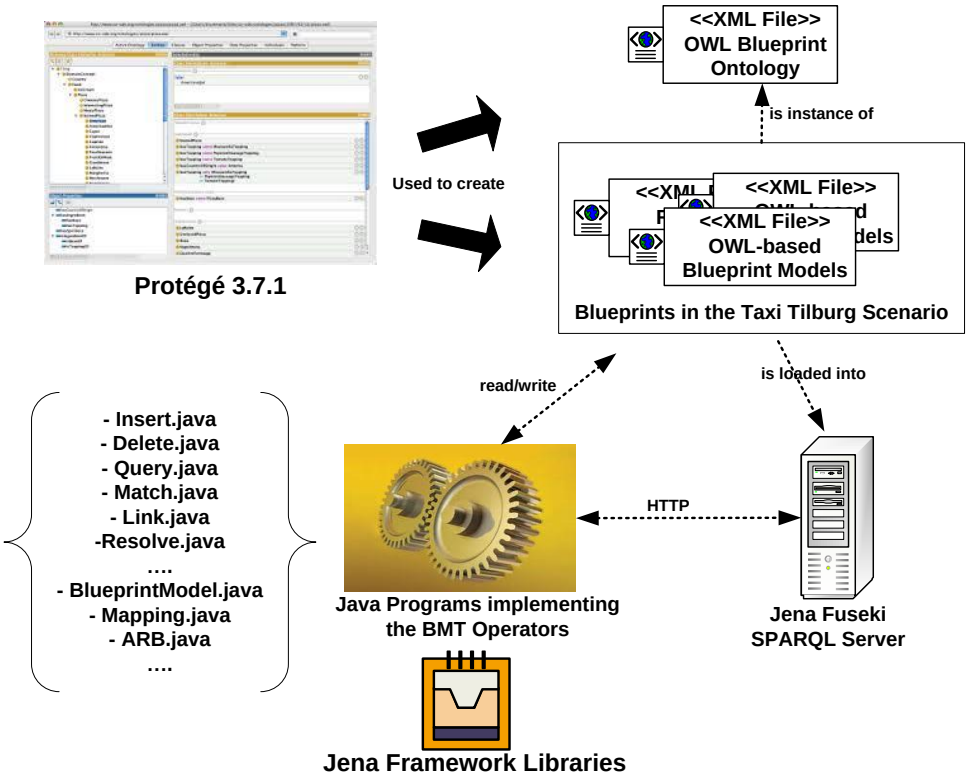
In this section, we aim to validate the feasibility of realizing the BSL and BMT's through a proof-of-concept prototype. Towards this goal, the *Taxi Tilburg Scenario* introduced in the previous Section 3.1 will be reused for the implementation. The following subsection 6.1.1 introduces the underlying technologies and tools that are used to build this prototype. Subsection 6.1.2 presents an overview of the prototype architecture. Then, the functionalities provided by the prototype are explained in subsection 6.1.3. Some experiments with the prototype are reported in subsection 6.1.4 and subsequently our findings are discussed in subsection 6.1.5. Lastly, subsection 6.1.6 summarizes our activity in validating the technical feasibility of the Blueprint Approach.

6.1.1 Underlying Technologies and Tools

The BSL has been formalized in the previous Chapter 4 as a set of OWL blueprint ontologies, and a blueprint model can be created by instantiating the OWL classes defined in these OWL blueprint ontologies. We use the well-known tool named *Protege*[1] for modeling the OWL blueprint ontologies. The Protege tool also supports the creation of a blueprint model as an instance of the OWL blueprint ontologies. A blueprint model is also described in OWL, hence it is called an OWL-based blueprint model. Other features supported by the Protege tool include the reasoning mechanism to ensure the consistency of an OWL-based blueprint model against the OWL blueprint on-

[1]Protege tool: `http://protege.stanford.edu/`

Figure 6.1: The proof-of-concept prototype for the Blueprint Approach



tologies, and the serialization of both the OWL blueprint ontologies and OWL-based blueprint models into XML documents. As a summary, we rely on the Protege tool to formally develop the BSL as a set of OWL Blueprint Ontologies and then create some sample OWL-based blueprint models.

The Jena Framework Libraries[2] is a well-known open-source Java framework for building Semantic Web applications. It supports a Java API for manipulating RDF data, an ontology for manipulating OWL and RDFS ontologies, and a set of other rule-based inferring and querying engines for RDF and OWL data sources. Given the various useful supports of the Jena framework we decided to use it for the purpose of implementing the BMT's operators in Java.

6.1.2 Architecture

Figure 6.1 illustrates the overall architecture of the prototype. The upper part of the figure explains the use of the Protege tool to create several OWL-based blueprint models for the *Taxi Tilburg Scenario*. Please note that the tool Protege also supports the consistency checking of the instantiated OWL-based blueprint models against the OWL

[2]Jena Framework: `http://jena.apache.org/`

blueprint ontologies and the serialization of both the OWL blueprint ontologies and the OWL-based blueprint models into XML files.

The lower part of Figure 6.1 illustrates the architecture of our proof-of-concept implementation for the BMT operators. The operators have been implemented as Java Programs that reuse the Jena Framework Libraries[3] for the following purposes:

- Parsing an XML document into a RDF model (a Java object implementing the Jena's **Model** interface).
- Manipulating RDF resources, RDF properties, RDF literals, etc. in a RDF Model.
- Executing SPARQL queries and SPARQL-Update statements against a RDF Graph Store.
- Serializing a RDF Model into an XML file.

We use the Jena Fuseki server[4] as the implementation of a RDF graph Store that can process SPARQL queries and SPARQL-Update statements. The Java programs that involve the execution of SPARQL queries and SPARQL-Update statements are considered as clients of the Jena Fuseki server. The server is started on localhost and then it can load the sample OWL-based blueprint models in the *Taxi Tilburg Scenario* into its RDF Graph Store.

6.1.3 Functionality

This section summarizes the functionalities offered by the prototype:

- Using the provided Blueprint Ontologies modeled in Protege, the users can instantiate new blueprints to specify their cloud services on any layer of the cloud stack. The users are also supported with the ontologies for modelling the policy and resource description of their cloud services.
- Using the Protege tool, the users can verify whether their blueprints are consistent with the Blueprint Ontologies
- By executing the *Insert.java* (or *Delete.java*) program with a blueprint as input, the user can upload a blueprint to (or delete a blueprint from) the Jena Fuseki server.
- By specifying a query in the SPARQL language and inputing this query into the *Query.java* program, the user can query the needed blueprints from the Jena Fuseki server.

[3]Jena Framework: `http://jena.apache.org/`

[4]Jena Fuseki: `http://jena.apache.org/documentation/serving_data/`

- By executing the *Resolve.java* program for an input blueprint, the user can run an automatic composition of blueprints. This program reuses the *Match*, *Link*, and *Query* programs to retrieve the needed blueprints from the Jena Fuseki server, performs matchmakings between their offerings and requirements, and finally composes them into the final result.

6.1.4 Validation Experiment

For the purpose of validating the technical feasibility of the blueprint approach, we performed the following experiments on our prototype:

1. As the first step, the needed blueprints for the *Taxi Tilburg Scenario* were modeled using the Protege tool. The `TaxiOrdering-CSBA` blueprint was considered as the target blueprint that needs to be resolved and the other blueprints were considered as the source blueprints.

2. The *Insert* operator (implemented by the Insert.java program) was used to upload the source blueprints to the Jena Fuseki server.

3. The *Query* operator was executed with different (complex) query expressions defined in SPARQL. In particular:

   - First, the source blueprints were queried based on their functional offering, e.g. querying for "Database", "Context-as-a-service" blueprints, etc.
   - Then, the source blueprints were also queried based on both their functional offering and policy and resource properties, e.g. querying for a "Database" blueprint that offers more than 2 TB.

4. As the last experiment step, the `TaxiOrdering-CSBA` was resolved by executing the *Resolve.java* program. The result was an OWL/XML file containing two alternative compositions of bluerpints.

6.1.5 Findings and Discussion

In general, the experiment results have helped us confirm the intended functionalities of the prototype. However, we also encounter the following issues that need to be considered in the future work:

- Using the Protege tool ensures the consistency of the instantiated OWL-based blueprints. However, for some reasons the user may need to modify his blueprint without using the tool. This leads to inconsistency between his blueprints and the schema defined in the Blueprint Ontologies. At the moment,

there is no support for verifying the consistency of a blueprint outside the Protege tool. This issue points to the need to develop a BMT operator to verify the consistency of a blueprint.

- Currently, querying blueprints is supported with query expressions described in SPARQL. Although SPARQL is already a powerful query language for RDF data, it is quite a technical language and not very popular for users outside the semantic web community. We are thinking of supporting a more generic, user-friendly blueprint query language in the future work.

- Regarding the resolution functionality, we observe the following issues:

  - We were not able to justify the performance of the resolution due to the small scope of the validation scenario. In the future work, we expect to perform measurements on the response time and reliability of the resolution functionality. At this moment, only the expected behavior of this functionality can be confirmed.
  - We see the possibility to limit the solution space of the resolution process by involving the user's decision in each incremental resolution step. As we have seen in Section 5.9, the resolution process is an iterative process to resolve the requirements of the blueprints. If the resolution process can be broken down to each iteration, in which the user is able to select a subset of the alternative results for the next iteration, the solution space is smaller and the performance will be increased. This issue will also be taken into account in our future development of the resolution functionality.

6.1.6 Summary

For the purpose of validating the technical feasibility of the Blueprint Approach, we used the Web Ontology Language (OWL), the well-established standard language for specifying semantic web resources, to formalize the semantics of the BSL model. Then, we also use OWL to create sample blueprint models as the instances of the BSL model to specify cloud services in the *Taxi Tilburg Scenario*. To be noticed, this scenario is a simplified version of a 4caast demonstration scenario, which has been co-developed with several industry partners for validation purposes. For demonstrating the functionality of the BMT operators, we have developed them as Java programs that reuse the Jena Framework APIs. The sample OWL-based blueprint models in the *Taxi Tilburg Scenario* are treated as RDF Models in these programs, so that they can be manipulated and composed. In general, our experiments in this section have shown that it is feasible to follow this path of implementation to realize a complete toolset for the BSL and the BMT's, although some of the important findings should be taken into account

in the next development phase to improve the consistency and performance of the approach.

6.2 Practical Validity of the Blueprint Approach

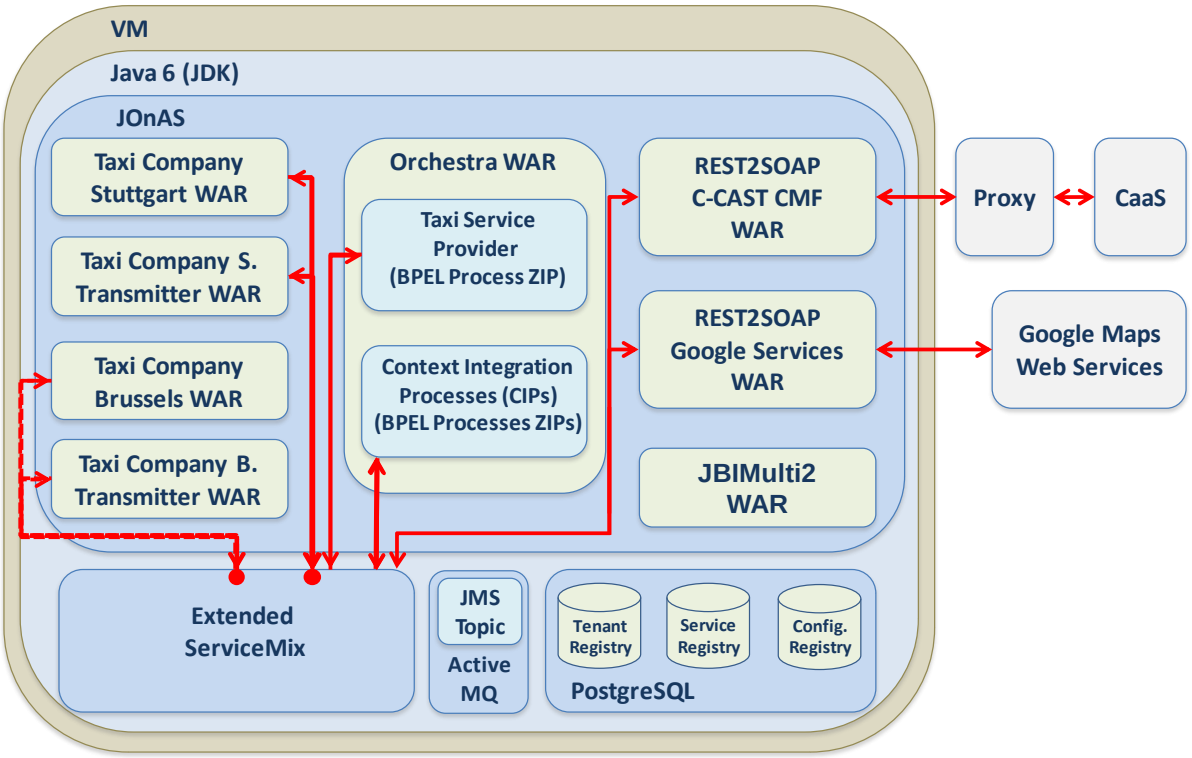
This section reports our activities in validating the applicability of the Blueprint Approach through our participation in the EC's 4caaSt FP7 project [European Comission, 2010]. From the early phase of this project, the blueprint approach has already been recognized as one of its main innovations [Gómez et al., 2012]. Its development started with a collaborative case study design with our industry partners in the 4caaSt project such as Telefonica, TelecomItalia, SAP, etc., which resulted in a set of 4caaSt's demonstration scenarios. The 4caaSt scenarios have been used throughout the four-year project term for validating the research results in general and the blueprint approach in particular. We introduce in the following subsection 6.2.1 the *Taxi Application Scenario* that has extensively been used during the course of the project for validating the Blueprint Approach. The prototype architecture implementing this scenario will then be introduced in subsection 6.2.2. Subsection 6.2.3 demonstrates how the components of this prototype architecture are modeled in a set of blueprints and how the blueprints are resolved to form a complete cloud application for the scenario. As part of the prototype, the blueprint toolset is introduced in subsection 6.2.4 to support the management and composition of blueprints. Finally, evaluation results of the Blueprint Approach are reported in subsection 6.2.5 and 6.2.6. Whilst the subsection 6.2.5 presents the evaluation result only within the 4CaaSt community, in subsection 6.2.6 we report the results of an more general questionnaire that has been tailored towards a group of knowledgeable audience in the SOA and Cloud Computing domain.

6.2.1 4caaSt's Taxi Application Scenario

Among the three validation scenarios in the 4CaaSt project, the benefit of using blueprints for cloud application development has been emphasized at most in the *Taxi Application Scenario* [Gómez et al., 2012]. This scenario has the objective to evaluate several features and technologies provided by the 4CaaSt project. For the full details of the scenario please refer to the description in [Gómez et al., 2012]. In the following we summarize the scenario to give the readers an overview of the validation context.

The *Taxi Application Scenario* is based on a hypothetical software company called **SaaSSolutionS** that develops a SaaS application for managing taxi fleets. There is another SaaS provider in this scenario called **FindMyWay.com** who offers a Context-as-a-Service (CaaS) that provides context information in real or near-real time. The 4CaaSt platform and 4CaaSt marketplace developed by the 4CaaSt project participate

Figure 6.2: Architecture Overview of the Taxi Application Prototype [Andrikopoulos et al., 2013]



in this scenario as the offerings of a stakeholder called **4CaaSt** who plays both the roles as a PaaS provider and as a marketplace. Finally the scenario involves two Small-and-Medium-Enterprise (SME) customers called **Taxi Company Stuttgart** and **Taxi Company Brussels**, who own their fleets of radio taxis.

The scenario starts when one of the two customer SME begins to browse the 4CaaSt marketplace for a Taxi fleet management software that meets their expectations. After finding the SaaS provided by SaaSSolutions they decide to contract this solution with the 4CaaSt marketplace. This contracting activity triggers the 4CaaSt marketplace to resolve both the technical and business requirements of this SaaS. With the CaaS solution provided by FindMyWay.com and the platform solutions provided by the 4CaaSt platform, it is able to resolve all the requirements of the taxi fleet management software. Lastly, the taxi fleet management software is deployed on the 4CaaSt platform and becomes ready-to-use for the Taxi Company Stuttgart and Taxi Company Brussels.

6.2.2 Taxi Application Architecture

In this section, we explain the architecture of the taxi application prototype that has been developed by our partners in the 4CaaSt project. Figure 6.2 illustrates the components that are parts of the deployed prototype, which in-

clude [Andrikopoulos et al., 2013]:

- *Taxi Application*: This component is provided by **SaaSSolutions** to implement the taxi management application for both the **Taxi Company Stuttgart** and **Taxi Company Brussels**. It contains the following software artefacts: Taxi Company Stuttgart WAR, Taxi Company Stuttgart Transmitter WAR, Taxi Company Brussels WAR, Taxi Company Brussels Transmitter WAR, and Taxi Service Provider (BPEL Process ZIP). The web-based front-end of the *Taxi Application* is implemented by the WAR files and its back-end is implemented as a set of BPEL processes.

- *Context Integration Framework*: This component is provided by **SaaSSolutions** to integrate with an external context information provider and an external map provider. It contains the following software artefacts: Context Integration Processes (CIPs), REST2SOAP CaaS CMF WAR, and REST2SOAP Google Services WAR.

- *Enteprise Service Bus (ESB)*: This component is provided by **SaaSSolutions** to enable the message communication between the *Taxi Application* component and the *Context Information Framework* component. It contains the following software artefacts: Extended ServiceMix, Active MQ, and JBIMulti2 WAR.[5].

- *JOnaS*, *Orchestra* and *PostgreSQL*: These are the components provided by the **4CaaSt platform**. *JOnaS*[6] is an open-source Servlet container to deploy the WAR files, *Orchestra*[7] is an open-source BPEL engine for hosting BPEL processes, and *PostgreSQL*[8] is an open source database solution.

- *CaaS*: The SaaS offered by **FindMyWay.com** to provide context information at real-time.

- *Google Maps Service*: the map service of Google.

6.2.3 Blueprint and Blueprint Resolution

For the design, configuration and deployment of the *Taxi Application scenario*, blueprint has been used as the uniform specification of (a group of) components of this prototype. Each blueprint specifies the requirements, offerings and artifacts needed for deploying the solution.

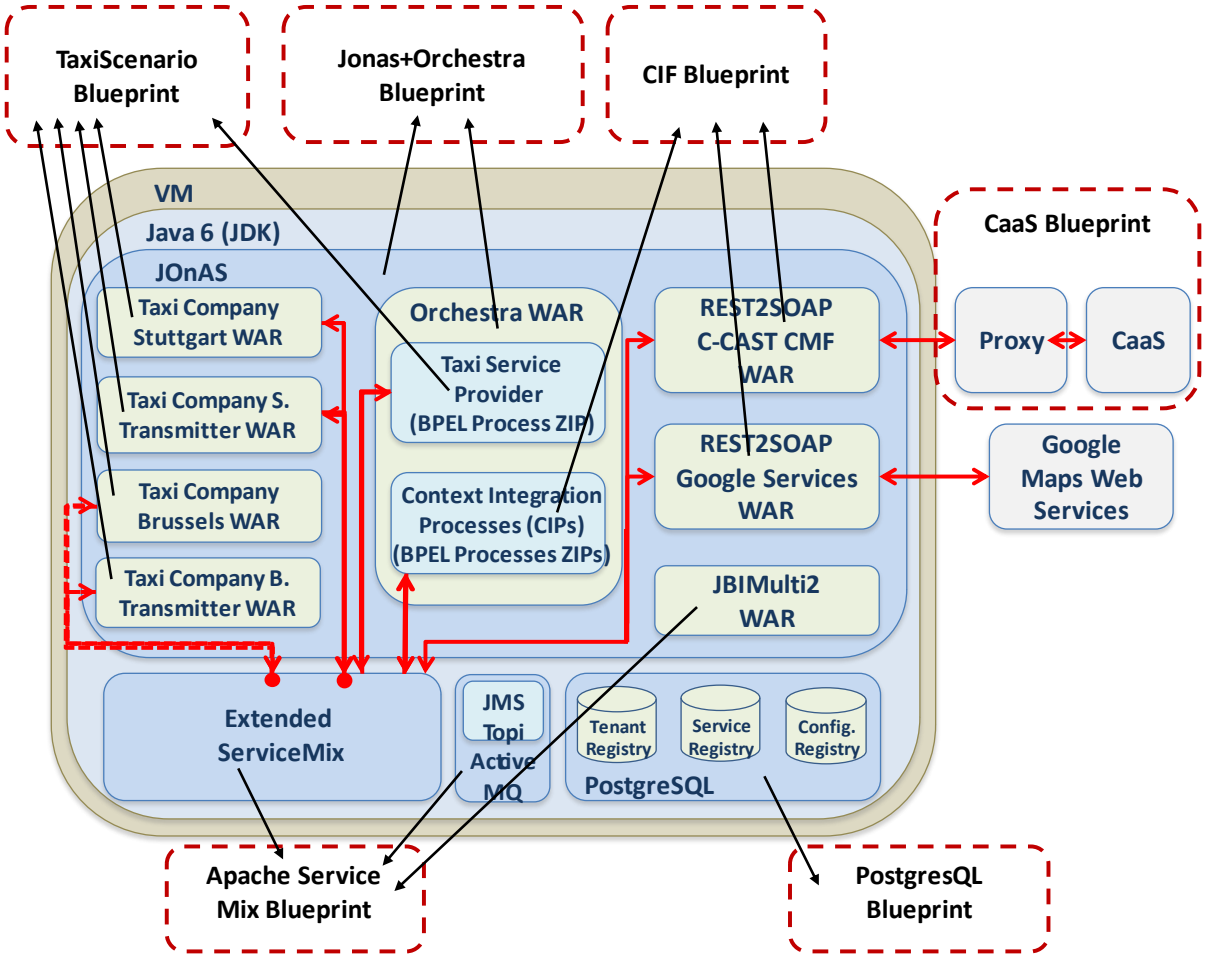
[5]This ESB component is actually provided by Apache ServiceMix (`http://servicemix.apache.org/`), an open source solution for implementing an ESB

[6]`http://jonas.ow2.org/xwiki/bin/view/Main/`

[7]`http://orchestra.ow2.org/xwiki/bin/view/Main/WebHome`

[8]`http://www.postgresql.org/`

Figure 6.3: Components of the 4CaaSt Taxi Application Scenario that have been modeled in the blueprints

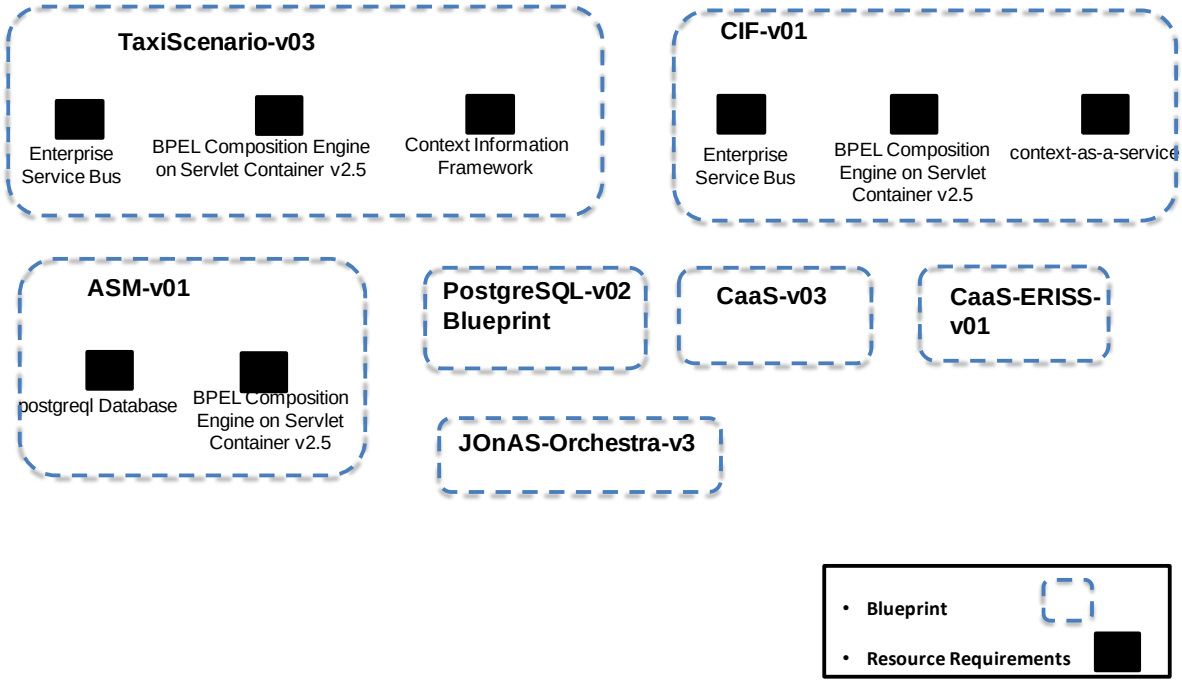


In the following subsection 6.2.3.1, we report our activities in co-developing the Blueprint Specification Language(BSL) with our partners in the 4CaaSt project and subsequently explain in detail the blueprints that have been created based on this language for the Taxi Application Scenario. In Subsection 6.2.3.2, we explain how to perform the blueprint resolution in the Taxi Application Scenario.

6.2.3.1 Blueprint Specification

A number of desired features for the blueprint definition have been derived from the need of a standardized cloud service description format in 4caaSt. We have organized several online and face-to-face meetings to gather these desired features from our industry partners. Then, we implemented the BSL in an XSD Template and distributed it to the 4caaSt community to get feedbacks. We organized a "blueprint training" virtual workshop to explain the concept of blueprint and train our industry partners how to use the XSD blueprint template. As the cloud service offerings identified in the 4caaSt

Figure 6.4: Blueprints of the 4Caast Taxi Application Scenario



scenarios are the real industrial offerings of our partners, we requested them to use the provided XSD blueprint template to describe their blueprints in XML and submit them to our blueprint repository. A set of XML-based blueprints have been created by our industry partners for the 4caast Taxi Application Scenario.

From the feedbacks of our partners during the collaborative development of the BSL, it has shown that the Blueprint XSD Template is capable to capture all the necessary aspects of an industrial cloud service offering and simple enough to be used by our industry partners. Following our guidelines during the blueprint training workshop, a TelecomItalia representative was able to design a blueprint "on-the-fly" for his cloud service. We experienced just a little amount of emails exchanged for the blueprint template support, which confirms its simplicity of use.

Figure 6.4 illustrates the blueprints that have been created by our 4CaaSt partners for the Taxi Application Scenario and how the components of the taxi application are captured in the blueprints. The list of the these blueprints is described in the following [Andrikopoulos et al., 2013]:

- *TaxiScenario-v03*: This is the blueprint describing the Taxi Application. It provides a "Taxi Management SaaS" and has three requirements: an "Enterprise Service Bus", a "Context Information Framework", and a "BPEL Composition Engine on Servlet Container v2.5". The first and second requirements represent the two functionalities needed by the offering and the last requirement indicates a platform component needed to deploy the war files and zip files, which are the

artefacts of this blueprint.

- *CIF-v01*: This is the blueprint that provides a "Context Information Framework" (CIF) product. It contains a number of artefacts that implement the CIF product. It exposes also three requirements needed for deploying and configuring the artefacts: a "BPEL Composition Engine on Servlet Container v2.5" , a "Context-as-a-Service", and an "Enterprise Service Bus".

- *JOnAS-Orchestra-v3*: This blueprint provides an integrated product containing an Orchestra BPEL composition engine that has been preconfigured and predeployed on a JOnAS application server. The JOnAS application server supports also a servlet container that can host servlet v2.5. Hence, this blueprint can be classified as a "BPEL Composition Engine on Servlet Container v2.5". This blueprint is ready for installation, thus contains no further requirements.

- *ASM-v01*: This blueprint provides the Apache Service Mix product. This type of offering can be classified as a "Enterprise Service Bus". The blueprint contains a number of artefacts that implement the product and the following two requirements: a "BPEL Composition Engine on Servlet Container v2.5" and a "postgresql database".

- *PostgreSQL-v02*: This blueprint provides a "postgresql database" product. The blueprint is ready for installation and thus contains no further requirements.

- *CaaS-BP-v03*: This blueprint provides a "Context-as-a-Service" product. The blueprint is ready for installation and thus contains no further requirements.

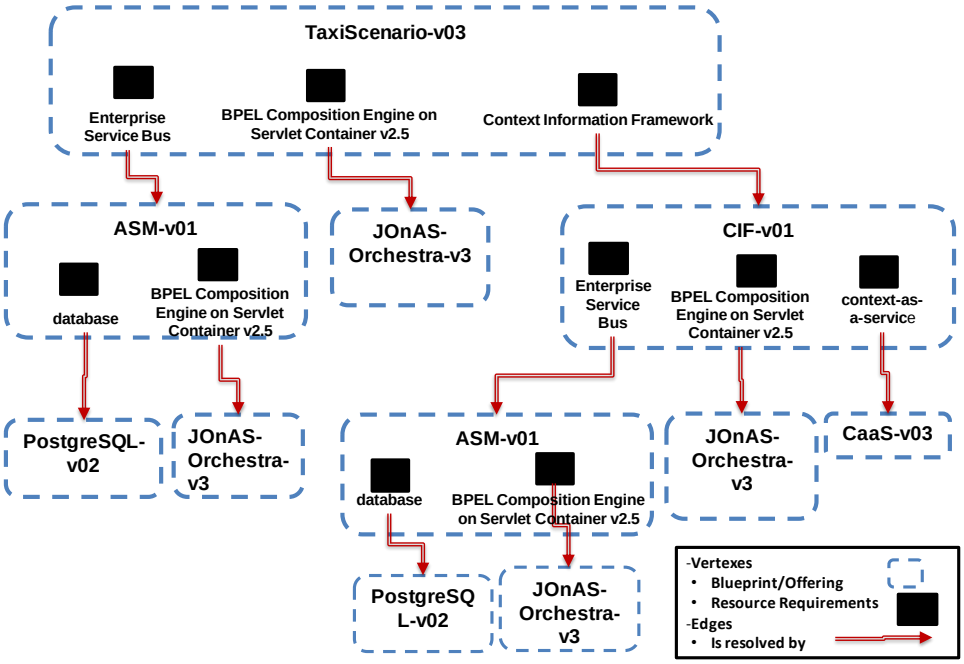
- *CaaS-ERISS-v01*: Similar to the CaaS-BP-v03 blueprint, this is another alternative blueprint which also provides a "Context-as-a-Service" product. The blueprint is provided by a different provider in the marketplace. It is also ready for installation and contains no requirements.

Figure 6.4 shows an intuitive illustration of all seven aforementioned blueprints that are involved in the Taxi Application scenario. Please note that, for the sake of brevity, only the blueprint name and the requirements of each blueprint are shown in the figure.

6.2.3.2 Blueprint Resolution

During the contracting of the Taxi Application, the requirements of the TaxiScenario-v03 blueprint are resolved to offerings of the source blueprints that are stored in the blueprint repository. The resolution is performed by following an algorithm that for each distinct solution creates the abstract resolved blueprint (ARB) for the Taxi application with all of the dependency blueprints embedded. Figure 6.5 and Figure 6.6

Figure 6.5: Candidate Abstract Resolved Blueprint (ARB) 1



show the two possible solutions for the resolution process. They differ in the choice of the Context-as-a-Service component, which is required by the CIF-v01 blueprint. Both solutions fulfill all of the requirements specified by the TaxiScenario-v03 blueprint and represent a physically deployable architecture.

Figure 6.6: Candidate Abstract Resolved Blueprint (ARB) 2

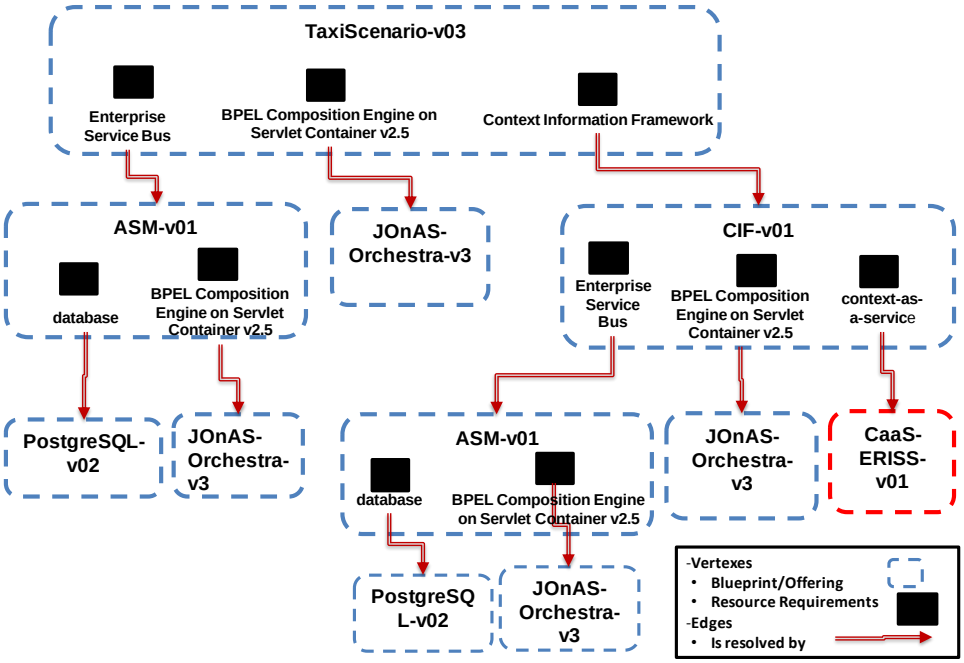
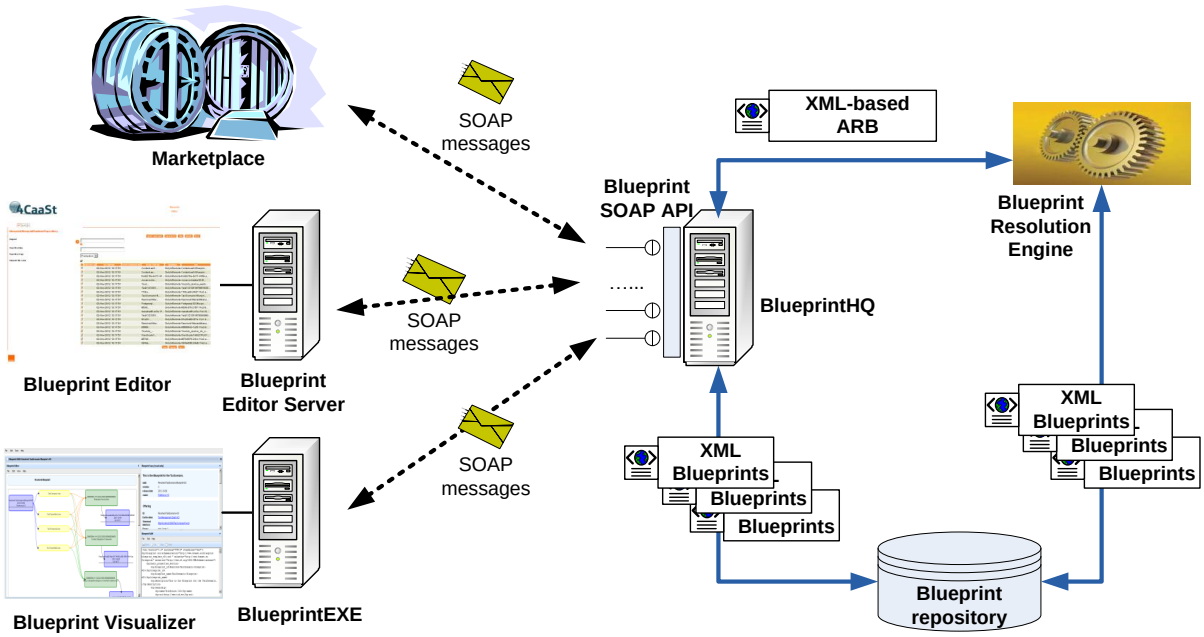


Figure 6.7: Architecture of the 4caaSt Blueprint Toolset



Note that the "Context-as-a-service (CaaS)" requirement is a part of the CIF-v01 blueprint, which is resolved recursively after the CIF-v01 blueprint has already been used to resolve the "Context Information Framework" requirement of the TaxiScenario-v03 blueprint. The two resolved solutions are passed to the marketplace in order to select the one that best fits the customer's preferences. Finally, the deployment manager for a physical deployment of the target service configuration is invoked with the chosen solution.

To understand how the blueprints in the Taxi Application Scenario are managed and composed by the resolution process, the next section 6.2.4 introduces the Blueprint Toolset that supports these tasks.

6.2.4 Blueprint Toolset Support

The blueprint toolset has been built with aim to validate the use of the BMT operators in the practice. The toolset has been extensively used by all three 4CaaSt demonstration scenarios for this purpose. It supports a number of tools for managing and manipulating XML-based blueprints in a repository, as well as for composing blueprints into a deployable CSBA configuration. Figure 6.7 depicts the architecture of the blueprint toolset, which contains the following components:

- Blueprint repository: is where all the XML-based blueprints in the 4caaSt project are stored. Currently, it is implemented as a MySQL database storing the XML files.

- BlueprintHQ (Blueprint Headquarter): is a Web Service providing a SOAP API that can be used by external components for accessing the blueprint repository and the resolution engine via SOAP message communication. The SOAP API contains the following SOAP operations:

  - addBlueprint(String blueprintXML): is the implementation of the BMT operator *Insert*. It accepts the entire XML-based blueprint as input and inserts this blueprint into the blueprint repository. A fault SOAP message is returned if there already exists another blueprint in the repository with the same ID.
  - deleteBlueprint(String blueprintID): is the implementation of the BMT operator *Delete*. It accepts the blueprint ID as input, identifies the blueprint in the repository with this ID, and finally deletes this blueprint out of the blueprint repository. A fault SOAP message is returned if there exists no blueprint in the repository with the input blueprintID.
  - getBlueprint(String blueprintID) and getAllBlueprints(): is the implementation of the BMT operator *Query* to query a blueprint from the repository based on its unique blueprintID, or to query all the existing blueprints in the repository.
  - resolveBlueprint(String blueprintXML): is the implementation of the BMT operator *Resolve*. Input of this operation is an entire XML-based blueprint. The BlueprintHQ will invoke the *Blueprint Resolution Engine*, which is another component of the toolset, with the input XML-based blueprint. The actual execution of the *Resolve* operator takes place in the blueprint resolution engine. For every invocation, the BlueprintHQ receives a resolution result (called Abstract Resolved Blueprint(ARB)). After all the ARBs have been retrieved, the BlueprintHQ combines them into a list of ARBs and returns the list as the output of the operation.

  The BlueprintHQ's SOAP API has been implemented using NuSOAP[9], a SOAP Toolkit for PHP. Its WSDL can be accessed under: `http://blueprinthq.host-for.me/4CaaSt/blueprint_hq.php?wsdl`. We have also built a test client for the BlueprintHQ under `http://blueprintexe.host-for.me/4CaaSt/blueprintHQ.htm#` for testing purpose. This SOAP API is consumed by (1) the marketplace component provided by our 4caaSt partners Telefonica and SAP to productize a blueprint and advertise it in a marketplace, (2) the Blueprint Editor provided by our 4caaSt partner FranceTelecom to enable the editing and managing of blueprints, and (3) our BlueprintEXE server that supports a web-based user interface for visualization purpose.

[9]NuSOAP: `http://sourceforge.net/projects/nusoap/`

- Blueprint resolution engine: is the engine that actually performs the resolution of an input XML-blueprint by iteratively performing the matchmaking of its requirements against the existing blueprints in the blueprint repository and finally producing a set of alternative Abstract Resolved Blueprints (ARBs) as the result. Each ARB is a possible combination of blueprints that constitute a cloud application. During the course of the 4caaSt project, we have worked with the folowing two implementations of the blueprint resolution engine:

  - Ericsson composition engine: During the first three years of the 4caaSt project, the Ericsson composition engine [Niemoeller et al., 2009] has been enhanced with the implementation of the blueprint resolution so that it could function as a blueprint resolution engine in our 4caaSt blueprint toolset. The midterm demonstration of the 4caaSt project has shown that the Ericsson composition engine is capable of delivering the expected resolution functionality. However, its performance still needs a lot of improvements.

  - IMDEA resolver: Recently, we have collaborated with the IMDEA Software Institute[10] in the co-development of the IMDEA resolver, which is an alternative blueprint resolution engine that aims to replace the Ericsson's composition engine with a better performance as well as more advanced features. The first release of the IMDEA resolver has met all the expectations regarding the performance of the blueprint resolution. By testing the IMDEA resolver with a 4caaSt scenario that involves seven source blueprint models in the repository and a target blueprint model, the time to get a solution is quite acceptable (about 332-662 ms) depending on the network conditions.

    The IMDEA resolver is a Java application that provides a Web service (with SOAP/HTTP messaging) and runs on top of a Glassfish 3.1+ application server. The main Java technologies involved in the implementation are Java XML Binding (JAXB) and Java Web Services (JAX-WS), of which Glassfish and its Metro component are the reference implementations.

- Blueprint Editor: is a Web interface that allows end-users to interact with the blueprint repository, e.g. to add, modify, display, or delete blueprints. It is currently provided and maintained by FranceTelecom, our partner in the 4caaSt project.

- Blueprint Visualizer: is a Web-based tool to visualize and resolve a blueprint. Hosted on the BlueprintEXE server, this tool communicates with the BlueprintHQ via SOAP messages to retrieve blueprints from the repository for

[10]IMDEA Software Institute: `http://software.imdea.org/about.html`

visualization purpose. Furthermore, it can also be used to trigger the resolution of a blueprint at the resolution engine.

  - The Blueprint Visualizer is accessible here: `http://blueprintexe.host-for.me/4CaaSt/blueprint_sdk_main.php`
  - A tutorial video showing how to use the Blueprint Visualizer can be seen here: `http://youtu.be/7UX5NfvP-Mw?hd=1`

- Marketplace: This component consumes the SOAP APIs of the BlueprintHQ service, with aim to retrieve blueprints and expose them as products to the customers. As long as a target blueprint is selected by the customer, the marketplace will trigger the resolution of the target blueprint in order to form the complete configuration for its deployment environment. Currently, the marketplace component is provided and maintained by Telefonica and SAP, our partners in the 4caaSt project.

The following components of the toolset: BlueprintEXE, BlueprintHQ, blueprint repository and blueprint resolution engine, are currently hosted on Flexiscale virtual machines[11] (supported by our 4caaSt partner Flexiant).

By using the blueprint toolset to demonstrate the manipulation and composition of blueprints in the Taxi Application Scenario, we have shown the feasibility of implementing a complete industry toolset for the BMT operators. We have particularly focused on the support for the design and configuration phase of the Taxi Application. Additional components supporting the runtime phase, e.g. the deployment controller, monitoring component, adaptation engine, etc., can easily be integrated to the toolset in the future via standard SOAP communication.

6.2.5 Evaluating the Blueprint Approach within the 4CaaSt Community

As described in the 4CaaSt Whitepaper [Gómez et al., 2012], the Blueprint Approach is one of the core contributions of the 4CaaSt project. Hence, it is crucial to evaluate its practical validity after demonstrating it in the Taxi Application Scenario. An evaluation questionnaire [Arozarena et al., 2012] has been created by the 4CaaSt project team and sent to the partners who have participated in the process of creating blueprints for the Taxi Application Scenario. The purpose of this questionnaire is to assess the current applicability of the Blueprint Approach and provide improvement suggestions for the future work in this direction.

The evaluation criteria contained in this questionnaire were taken from that presents the S-Cube quality reference model, which include [Arozarena et al., 2012]:

[11] Flexiscale: `http://www.flexiscale.com/`

- *Learnability* (notation scale –, -, 0, +, ++): Easiness of the blueprint model understanding.
- *Effectiveness* (notation scale –, -, 0, +, ++): Accuracy and completeness of the blueprint that the user designs with respect to the user's application or technology.
- *Efficiency of use* (notation scale –, -, 0, +, ++): Effort and means put in designing and applying the blueprint.

Four 4CaaSt partners who have created the Taxi Application blueprint (TaxiScencario-v03), the Context-as-a-Service blueprint (CaaS-v03), the JOnAS/Orchestra blueprint (JOnAS-Orchestra-v3), and the Context Integration Framework blueprint (CIF-v01), have responded to this questionnaire with their assessments. In the following we summarize the evaluation result that has been reported in [Arozarena et al., 2012]:

- Regarding the *Learnability*: This criterion receives two "++" and two "0", which indicates a quite high appreciation of the users in learning and using the blueprints. General comments state that since the blueprint is designed as an XSD template, it can be easily and intuitively created without extra learning effort. The good documentation and examples have also received lots of appreciations.
- Regarding the *Effectiveness*: This criterion receives three "+" and one "0". The good score of this criterion confirms the accuracy and completeness of the current blueprint structure. The extension mechanism of the blueprint template has been acknowledged as a significant feature for integrating with existing languages. There is also an interesting comment stating that since the blueprint is currently only capable to capture only one cloud service, several blueprints would be needed for several variants of the same cloud service. It is likely clearer to have a separate blueprint for each service variant, yet very inefficient if the number of variants is high.
- Regarding the *Efficiency of use*: This criterion receives two "+" and two "0". The score of this criterion is fairly good. It indicates that the users can work easily with the current design of the blueprint template. The feedbacks of this criterion stated that it did not require many hours to create a blueprint. The neutral scores in this case only indicate that several feedback cycles were needed to create a stable structure definition in the current blueprint template version.

6.2.6 Evaluating the Blueprint Approach in a broader Scope

The previous subsection 6.2.5 reports an evaluation activity that has been specifically tailored towards the 4CaaSt partners who have contributed to the Taxi Application

Scenario demonstration. The result is somewhat useful for the future development of the Blueprint Approach since it reflects the assessment of those who have worked directly with the blueprints. It does, however, not represent a general and objective assessment of the practical validity of our approach.

For the purpose of evaluating the Blueprint Approach in a more objective way, we have approached a group of knowledgeable people in the SOA and Cloud Computing domains with a more general questionnaire. Some of the target audiences are already familiar with the concept of blueprints. Some other are not and thus, need more time to get acquainted to the approach through the related literature and tools. The questionnaire contains 10 questions and is divided into three parts:

- Part 1 - Understanding the Participant's Segmentation: in this part we would like to understand the knowledge background of the participant in the field of cloud computing as well as his/her work circumstance.
- Part 2 - Evaluating the Blueprint Approach: in this part we would like the participant to give his/her assessment on the practical validity of the Blueprint Approach.
- Part 3 - Improvement Suggestions: in this part we would like to collect the improvement suggestions for the future development of the Blueprint Approach.

Out of the 22 people who have been asked to participate in the questionnaire, 14 people have delivered a complete result. The details of these anwers can be found in Appendix B. In the following, we analyze the 14 complete answers.[12]

6.2.6.1 Part 1: Understanding the Participant's Segmentation

In this part we would like to understand the knowledge background of the participant in the field of cloud computing as well as his/her work circumstance.

[12]Please note that in the questionnaire we used different terminologies for the Blueprint Approach, i.e. we use "Blueprint XSD Template" to refer to a concrete syntax of the BSL, and "Blueprint Web Service Interface" to refer to a concrete implementation of the BMT operators. The purpose is to increase the understandability for the technical audience.

Q1. *To which extent are you familiar with cloud application development?*

- Low
- Medium
- High

Q2. *Your work is...*

- Technically-oriented in an Academic Organization
- Technically-oriented in an Industry Organization
- Business/Economic-oriented in an Academic Organization
- Business/Economic-oriented in an Industry Organization

Q3.*What is your intended usage of the blueprint approach?*

- For further internal R&D activities
- For commercial product and service development
- Both
- Others
- No intended usage at all

Figure 7.1 in the Appendix B presents the results of the three questions Q1, Q2, and Q3 about the background and work circumstance of the participants. The majority of the participants (11 out of 14) have an adequate knowledge on cloud application development and 3 of them are already experts in the cloud computing domain. The questionnaire has gained attention from both academic and industry organization where there are slightly more people from the academia (8) than from the industry (6). It is understandable that most of the participants have a technical background (10 out of 14) as the Blueprint Approach involves much technical knowledge. Lastly, what is also interesting is to understand the intended usage of the Blueprint Approach. The majority of the participants (11 out of 14) would like to adopt this approach for further internal Research & Development activities whilst 4 participants would like to apply it immediately for their commercial products and services. This information gives us the encouragement that the Blueprint Approach has actually gained much interest from the outside community.

6.2.6.2 Part2: Evaluating the Blueprint Approach

In this part we report the participants' assessments on the practical validity of the Blueprint Approach. The question Q4, Q5, Q6 and Q7 in the questionnaire aim to evaluate the current capability and design of both the Blueprint XSD Template and the Blueprint Web Service Interface.

Q4. *The Blueprint Template has been developed as an XML-based template to capture a complete specification of a cloud service. The current version of the Blueprint Template is able to...*

- *4a. help the cloud providers shape their cloud service specifications in a more structured and meaningful way.*
- *4b. serve as a uniform specification of cloud services that can be shared and composed among a number of developers and cloud providers.*
- *4c. provide a possibility for cross-linking cloud service specifications with aim to facilitate the design and configuration of composite cloud applications.*
- *4d. facilitate a more efficient discovery and selection of cloud services on a marketplace.*
- *4e. help reduce the effort of (re-)composing the cloud service in different composite cloud applications.*
- *4f. enable a more efficient way to aggregate prices from different pricing models as well as resolve the business conflicts.*
- *4g. support the specification of the architecture topology of an entire cloud application, from which an automatic deployment plan can be generated.*

The question Q4 is used to evaluate the current capability of the Blueprint Template. It has been designed as a rating scale question (1 = strongly disagree, 2 = disagree, 3 = neutral, 4= agree, 5 = strongly agree). The upper part of Figure 7.2 in the Appendix B visualizes the average scores of this question. All participants has agreed that the Blueprint Template has achieved its goal to help the cloud providers shape their cloud service specifications in a more structured and meaningful way (statement 4a). The other statements 4b, 4c, 4d, 4e and 4g have also received relatively good feedbacks. However, the statement 4f only receives an intermediate score, which clearly indicates that the blueprint template is still not able to support the aggregation of prices and resolution of business conflicts. This can be explained in the way that the current version of the blueprint template still does not take this information into account and relies on its extension mechanism for supporting the price specification, price aggregation and

business conflict resolution.

Q5. *The current design of the Blueprint Template...*

- *5a. has captured all the essential information of a cloud service specification.*
- *5b. is easy to understand and follow.*
- *5c. is logically divided into meaningful template sections, each focusing on a particular information set of a cloud service specification.*
- *5d. can easily be extended to incorporate external languages or specification schemas, e.g. for specifying monitoring and accounting information.*
- *5e. supports the interchangeability between heterogeneous systems thanks to its XML-based definition.*
- *5f. does not require additional training/guidance to understand and apply.*

The question Q5 is used to evaluate the current design of the Blueprint Template. It has been designed as a rating scale question (1 = strongly disagree, 2 = disagree, 3 = neutral, 4= agree, 5 = strongly agree). The lower part of Figure 7.2 in the Appendix B visualizes the average scores of this question. In general, the current design of the template is acceptable. The highest score was given to statement 5c stating that the template has been designed with a logical and understandable structure. From the result of this question, we recognize the two biggest concerns: (1) regarding the statement 5f, the template might be difficult for the users so that additional training and guidance is still much expected, and (2) regarding the statement 5a, the template might still be incomplete so that additional information elements of a cloud service specification need to be incorporated into the future version of the template.

Q6. *Currently, the Blueprint Template contains the following elements: Offering, Deployment Artefact(s), Requirement(s), Architecture Topology. What are the 3 most important elements that should (still) be supported in the next version of the Blueprint Template?*

- *6a. Offering*
- *6b. Deployment Artefacts*
- *6c. Requirements*
- *6d. Architecture Topology*

In the question Q6, we would like the participants to select the most important elements of the current blueprint template. The result of this question is visualized in the upper part of Figure 7.3 in the Appendix B. The aim of this question is to re-

vise the significance/validity of these elements for the future template version. The majority of participants (12 out of 14) agrees that "Offering" and "Requirement" are the inherent elements of the template. There are some doubts about the role of the "Deployment Artefacts" and "Architecture Topology" elements in the template. A logical explanation for it is that some participants may expect a clear boundary between the Blueprint Template (which aims to support the design and composition of cloud services) and other cloud specification formats like OVF [DMTF, b] and TOSCA [OASIS, 2013] (which aim to support the the configuration of the deployment environment of cloud services)

Q7. *The Blueprint Web Service Interface has been developed to support a set of functionalities for creating, manipulating, and composing blueprints. The current version of the Blueprint Web Service Interface is able to...*

- *7a. provide the essential functionalities in managing and composing cloud services.*
- *7b. (through the Resolve operation) facilitate the automatic discovery and composition of cloud service specifications to meet certain application's requirements.*
- *7c. deliver reliable results.*
- *7d. deliver a reasonable performance.*
- *7e. easy to integrate with external systems.*

The question Q7 aims to evaluate the current capabilities of the Blueprint Web Service Interface, which is a Web Service-based implementation of the BMT operators. The average scores of this question are visualized in the lower part of Figure 7.3 in the Appendix B. Although this question has received reasonably good scores, the biggest concern remains in the completeness of the provided functionalities (statement 7a). The evaluatio result of the statement 7a is totally in-line with a large number of suggested functionalities for the next version of the Blueprint Web Service Interface that will be reported in the next Question Q8. Furthermore, one of the significant remarks in the result of this question is that the "Resolve" operation has gained a lot of appreciations as a significant functionality that supports the automatic discovery and composition of cloud service specifications for engineering cloud applications.

6.2.6.3 Part 3: Improvement Suggestions

The third part of the questionnaire contains three questions that ask for the participants' recommendations to improve the Blueprint Approach in the future work.

Q8. *In your opinion, what are the 3 most important functionalities that are currently missing in the Blueprint Web Service Interface ?*

The question Q8 is an open question asking for the suggested functionalities for the Blueprint Web Service Interface. From the result of the previous question Q7 we already see that the participants also have some doubts about the completeness of the provided functionalities. For this question, a list of desired functionalities have been provided, which will be discussed and evaluated in the following regarding their significance and implementability (notation scale –, -, 0, +, ++):

- Listing and grouping blueprints: There is a desire of grouping (tagging) blueprints so that they can be organized into meaningful groups (not necessarily hierarchically). This desire involves the definition of blueprint categories and a category-based management of blueprints in the repository. Significance (+), Implementability (+)

- Checking the compliance of regulatory (e.g. Sarbens-Oxley) and other compliance requirements against the end-to-end Blueprint model: This is indeed a legitimate recommendation since the composition of web services may have to face the situation in which several cloud services operate under different regulatory rules. However, this topic involves a large body of knowledge and may require much effort in the future work. Significance (++), Implementability (-)

- Rich querying of the blueprint repository: The simplicity of the current querying functionality is a known shortcoming of the Blueprint Approach. A more generic query operation together with a rich query language will definitely be developed in the next step to support the users with more complex query expressions. However, this desire also requires relatively much effort. Significance (++), Implementability (-).

- Automatic substitution of blueprints: We only consider this suggestion as a "nice-to-have" feature for the future work. Significance (0), Implementability (0)

- Generate Deployment Plan: This is an important suggestion. However, a number of related approaches like OVF [DMTF, b] and TOSCA [OASIS, 2013] have already been developed in this direction. Hence, we would like to combine these approaches with the Blueprint Approach and reuse their features to generate a deployment plan for the blueprints, rather than develop this feature on our own. Significance (-), Implementability (+)

- Semantic support for blueprints: Although our theoretical work on the blueprint has been based on semantic web technologies like RDF and OWL, we do not

intend to provide the semantics support for the users in the near future. Significance (-), Implementability (0).

- Check the XML syntax of new/updated blueprints: This is a valid suggestion since we would always like to ensure the consistency of the blueprints that are submitted to the repository. In our opinion, it is also feasible to implement the consistency checking functionality. Significance (+), Implementability (++)
- Support for an interactive resolution to allow users to provide their choices, knowledge in the process of resolving blueprints: This is an interesting suggestion to improve the performance and reliability of the resolution process. Significance (++), Implementability (0)

Q9. *In your opinion, what are the 3 most important language extensions for the Blueprint Template?*

- *9a. A language for specifying the Metering Information for a blueprint including e.g. a description of what needs to be measured (event, time, quantity, etc.), and the measurement unit (number of invocation, number of notification, millisecond, second, GB, MB, etc.).*
- *9b. A language for specifying the Monitoring Information for a blueprint including e.g. a description of what needs to be monitored (physical memory, current CPU load, bandwidth, etc.) and the monitoring unit (GByte, Gbit/s, CPU-Instruction, etc.)*
- *9c. A language for specifying the Constraints among several blueprints including e.g. the business or regulatory compliance rules between the blueprint providers.*
- *9d. A language for specifying the Deployment Environment of a blueprint including e.g. the properties of the virtual machine nodes and network links used to deploy the blueprint together with their vertical and horizontal elasticity.*
- *9e. A language for specifying the Pricing Information of a blueprint including e.g. the pricing model (pay-per-use for subscription-based), price unit, price aggregation strategy, etc.*
- *9f. A language for specifying the Service Level Agreements (SLA) of a blueprint including e.g. the required performance level of a cloud service, the penalty and compensation policy , etc.*

The Blueprint Template provides an extension mechanism which can be used to incorporate external languages or specification schemas. We use the question Q9 to ask for selecting the potential language extensions that play an important role in cloud ap-

plication development. The list of potential language extensions have been provided in the question. The result of this question is visualized in the upper part of Figure 7.4 in Appendix B. The most desired language extension is for specifying the Service-level Agreement (SLA) of a blueprint (statement 9f). Another important language extension is for specifying the constraints between the blueprints to support the specification of business or regulatory compliance rules between the blueprint providers (statement 9c). Other proposals for specifying the deployment environment, the pricing information, and the metering and monitoring information, also receive much attention from the participants.

Q10. *In your opinion, what are the 3 most important components that should be supported in the future development of the Blueprint Approach?*

- *10a. A software tool to generate and configure the deployment environment of a blueprint*
- *10b. A software tool to monitor the runtime SLA of a blueprint, e.g. to monitor the response time, throughput and availability of the cloud service specified in that blueprint*
- *10c. A software tool to monitor the resource consumption of a blueprint, e.g. to monitor the CPU load and the memory and bandwidth consumptions.*
- *10d. A visualization tool for blueprints and blueprint compositions that enables easy and intuitive editing tasks.*
- *10e. A software tool that helps manage the entire lifecycle of a blueprint, i.e. from the design and composition to the deployment, monitoring, and retirement.*

Currently, the Blueprint Approach supports (1) a Blueprint XSD template for specifying a cloud service in a blueprint, and (2) a Blueprint Web Service Interface for managing and composing blueprints. In the last question Q10, we would like to ask for recommendations for the additional tools that should be supported by the Blueprint Approach in the future. The lower part of Figure 7.4 in the Appendix B visualizes the result of this question. The majority of the participants (12 out of 14) points out the significance of having a visualization tool for blueprints and blueprint compositions that enables easy and intuitive editing tasks. The second most important tool support is a lifecycle management tool for the blueprints. Both of the two most desired tool supports are already in the list of our future developments. In particular, the beta-version of the visualization tool[13] has been introduced to the 4CaaSt community.

[13]The beta-version of the blueprint visualization tool is located under `http://blueprintexe.host-for.me/4CaaSt/blueprint_sdk_main.php`

CHAPTER 7

CONCLUSIONS AND FUTURE ISSUES

The convergence of Service-Oriented Architecture (SOA) and Cloud Computing introduces a novel approach to the engineering of distributed, geographically dispersed Service-based Applications (SBAs). Composing SBAs using a syndication of heterogeneous cloud-based software-as-a-service (SaaSs), platform-as-a-service (PaaSs) and infrastructure-as-a-service (IaaSs) allows an SBA developer to take advantage of their different functionalities and qualities to build a tailored solution that meets specific business requirements. SBAs that are built this way are called Cloud Service-Based Applications (CSBAs).

However, current approaches for engineering CSBAs usually lead to a vendor lock-in situation where the constituting SaaSs used to compose an CSBA are delivered as monolithic, one-size-fits-all solution stacks coupled to proprietary platforms and infrastructures. These monolithic SaaS stacks are usually not customizable, extendable and interoperable. To support a more flexible approach for engineering CSBAs, we propose in this thesis the concept of *Blueprint* as an abstract, uniform specification of a cloud service across all three layers of the cloud stack, i.e. SaaS, PaaS and IaaS. The **contribution** of this thesis is to provide a *Blueprint Approach* for engineering CSBAs that includes the following components:

- A well-defined *Blueprint Specification Language* (BSL) that provides a means for cloud service providers to abstractly (i.e., independent of the implementation) and unambiguously specify a cloud service in a blueprint. Chapter 4 has introduced the BSL as a Domain-specific Modelling Language that comprises of the following components: the BSL abstract syntax model in UML, the Blueprint XSD Template as the BSL concrete syntax, and the OWL blueprint ontologies as the formalization of the BSL semantics. Mappings between the three components

of the BSL have also been explained in this chapter.

- A set of *Blueprint Manipulation Techniques* (BMTs) for publishing, querying, and composing blueprints with aim to support the design and configuration of an CSBA. In Chapter 5, we have developed these techniques as the *BMT operators* that work on the blueprints. Each BMT operator has been introduced in detail including its conceptual definition, formalization and implementation.

Following the blueprint approach, CSBA developers can create sophisticated CSBAs from heterogeneous SaaS, PaaS, and IaaSs offered by different providers to achieve end-to-end business requirements. A sample CSBA has been used as a running example throughout the thesis to demonstrate our approach, i.e. the *TaxiScenario-CSBA*. It has also been implemented as a "proof-of-concept" prototype to prove the feasibility of the blueprint approach. To demonstrate the applicability of the blueprint approach in the cloud computing domain, a blueprint toolset has been co-developed with our industry partners within the scope of the 4caaSt project [European Comission, 2010].

Given the research results, we review in Section 7.1 the research questions of the thesis and provide the answer for each question. Then, in Section 7.2 we evaluate our results based on the criteria of design science. Finally, in Section 7.3 , we present the issues that can be developed in the future to tackle the limitations of our research.

7.1 Research Questions and Answers

In the previous Section 1.4, we have defined four distinct research questions for the thesis. In this section, we provide the answers to them.

RQ1: What is the *state-of-the-art* in specification language supports for cloud services and what are their strengths and shortcomings?

The state-of-the-art in cloud service specification languages has been extensively discussed in Chapter 2. We discovered that these languages are restricted to a certain cloud layer, e.g. SaaS, PaaS, or IaaS. On the IaaS layer there exists a large body of work on IaaS definition languages that aim to support an automatic packaging and deployment of IaaS solutions on an infrastructure cloud. On the PaaS layer, there exist only proprietary PaaS specification languages supported by particular vendors. On the SaaS layer, existing well-established standards in Web services and SOA services can certainly be reused for specifying SaaSs. We identified the lack of a uniform specification language for cloud services across all three layers of the cloud stack. As the result, we proposed the *Blueprint Approach* to address this shortcoming.

RQ2: How can we *design* and *validate* a uniform specification language for cloud services across all three layers of the cloud delivery stack?

Chapter 4 has proposed the BSL as the uniform specification language for cloud services across all three layers of the cloud delivery stack. The BSL is divided into modular modules to cover all aspects of a cloud service specification. Depending on the cloud layer, a user of the BSL can choose the appropriate modules to specify his cloud service, e.g. he can choose the BSL SaaS Module (which imports the BSL Core Module, BSL Policy Module and BSL Interface Description Module) to specify his SaaS, or he can choose the BSL PaaS module (which imports the BSL Core Module, BSL Policy Module, and BSL Resource Description Module) to specify his PaaS. Furthermore, the BSL is designed in an extensible way so that external languages and specification schemas can easily be incorporated.

A concrete XML-based syntax of the BSL has been proposed and the semantics of the BSL has also been formalized as an OWL schema model. The validation of the BSL has been performed through the distribution of the XML-based template to our industry partners in the 4caaSt project [European Comission, 2010] to gain feedbacks. The result has shown that the BSL has captured all the needed features of a uniform cloud service specification language, yet is still simple enough to be used by the industry people.

RQ3: How can we *design* and *validate* a set of manipulation techniques for publishing, querying, and composing cloud service specifications

Another goal of developing the *Blueprint Approach*, next to the *BSL*, is to develop a set of associated techniques that aim to support the publishing, querying and composition of cloud service specifications. By using the BSL, cloud services can be specified in blueprints. The *Blueprint Manipulation Techniques* (BMTs) have been developed in Chapter 5 as a set of BMT operators that aim to support the manipulation and composition of blueprints.

In Chapter 5 we have introduced a set of BMT Operators including their conceptual definitions, formalization, and pseudo implementation. To prove the feasibility of these operators, a proof-of-concept prototype has been built using the existing semantic web technologies in querying, updating and manipulating OWL/RDF data. We have also participated into the development of the 4caaSt blueprint toolset, which has been built based on the ideas of the BMT operators. The 4caaSt blueprint toolset is currently being used in several industry prototypes and attracts a lot of interests from the cloud market leaders like Telefonica, SAP, FranceTelecome, etc.

7.2 Evaluation

As introduced in the previous Section 1.5, the development of the *Blueprint Approach* conceptually belongs to design science. Hence, it is necessary to evaluate

the research result against the requirements for an effective design science research. In [Hevner et al., 2004], a list of criteria for evaluation has been introduced. In the following, we use these criteria for evaluating our research result:

Design as an Artifact: Our work delivers the following viable artefacts:

- The BSL as a Domain-specific Modeling Language including:
  - The BSL abstract syntax as an UML model
  - The BSL concrete syntax as an XSD template
  - The BSL formal semantics as a set of OWL ontologies.
- A set of BMT operators for manipulating and composing blueprints, including
  - Their conceptual definitions.
  - Their formalizations.
  - Their prototype implementations.

All the artefacts have been introduced with a running example.

Problem Relevance: In the previous Section 1.3, we have identified the problem definition of our research as the lack of a uniform cloud service specification language and a set of associated techniques to manipulate and compose cloud service specifications. The development of the *Blueprint Approach*, which comprises of the BSL and the BMTs, is totally in line with the problem definition.

Design Evaluation: The *Blueprint Approach* has been evaluated using an experimental method. More specifically:

- We opt to evaluate the applicability of our approach using a CSBA engineering scenario driven from an industrial case of the 4caaSt project. In Chapter 4 and 5, we have demonstrated how the BSL and BMTs are able to support the specification, manipulation, and composition of cloud services within this CSBA engineering scenario.
- Evaluation activities have been continuously performed within the 4caaSt community with aim to gain feedbacks on the utility, quality and applicability of the *Blueprint Approach*. In particular, we have organized several meeting and training sessions to motivate the use of our *Blueprint Approach* within the scope of the 4caaSt project. After the four-year term of the 4caaSt project, the *Blueprint Approach* has received many positive feedbacks from our industry partners on addressing current shortcomings in the CSBA engineering domain. It has also become one of the core contributions of the 4caaSt project.

Research Contributions: The contributions of this work have been discussed in detail at the beginning of this chapter.

Research Rigor: The semantics of the BSL has been formalized as a set of OWL ontologies. The reason for choosing the OWL language for formalizing the BSL is that it is a well-established standard language for knowledge representation. It has also been proved in [Horrocks et al., 2007] that OWL is based on the Description Logic, which is "a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way". The formalization of the BSL as a set of OWL ontologies would provide the sufficient level of formalism for our BSL contribution.

By instantiating the OWL classes defined in the OWL ontologies, blueprints can be formally specified as OWL models too. Since the OWL language extends the RDF vocabulary, an OWL-based blueprint model is also a RDF graph. We used the formal structure of a blueprint as a RDF graph to formalize the BMT operators that work on it. More specifically, the BMT operators have been formalized as either (1) SPARQL operations for querying RDF Graphs, or (2) SPARQL-Update for updating RDF graphs, or (3) basic graph operations for manipulating a graph data structure.

However, it is important to mention that given the innovative nature of our work in the domain of CSBA engineering, we would prefer to put focus on the applicability of the *BSL* and *BMTs*, rather than pursuing to prove their mathematical rigor.

Design as a Search Process: The development of the *Blueprint Approach* has been motivated by the lack of a uniform cloud service specification language and a set of associated manipulation techniques to support CSBA engineers and cloud service providers within the CSBA Engineering Lifecycle. Given this motivation, we have performed an extensive state-of-the-art analysis on the current language approaches. The result has shown that existing languages are typically restricted to a certain cloud layer and thus not completely suitable to be used in the CSBA engineering lifecycle. This has given us the motivation to develop the BSL. Moreover, the CSBA engineering lifecycle has also pointed out the need to develop the associated techniques to manipulate and compose cloud service specifications. This need has led to the development of the BMTs. Inspired by the idea of model management operators that have been developed in [Melnik, 2004] for manipulating data models, we have developed a set of operators to support the BMTs. Finally, to validate the feasibility and applicability of our solutions, we have built a proof-of-concept prototype as well as participated in several demonstration prototypes in the 4caaSt project. In summary, the development of the *Blueprint Approach* has really been a search process starting always with a problem definition, then searching for a solution among the existing state-of-the-arts, then building a solution, and finally validating that the solution we built is applicable.

Communication of the Research: The thesis has been presented in a way that is in-

teresting to both technical and non-technical audiences. Firstly in Chapter 1, high-level descriptions of the motivation, problem definition, goal and questions of the thesis have been introduced to motivate both the technical and non-technical audiences. Then, Chapter 3 has presented the core contributions of the thesis that are still understandable for non-technical audiences. More in-depth technical details about the BSL and BMTs have been presented in Chapter 4 and 5 that aim merely to the technical audiences.

7.3 Future Issues

Engineering CSBAs is a relatively new research domain and still contains many challenges that our *Blueprint Approach* cannot support. It is inevitable that our contributions still contain a number of limitations, which have already been explained in the previous section 1.7. In this section, we present the future work and directions, some of which also address the limitations:

- *Extensions of the BSL*: Currently, the BSL allows for specifying only limited information categories of a cloud service, e.g. its functionality, policy, resource, interface, etc. These categories have been selected as the most significant aspects of a cloud service specification from existing literature and from the discussions with our industry partners. However, several important aspects of a cloud service have been left out such as the pricing model, licensing, regulatory rules, etc. In order to have a complete picture of a cloud service specification, several extensions are required for the BSL in the future.

- *Extensions of the Policy and Resource Description Modules*: Currently the BSL lacks of an expressive policy and resource description for a cloud service. This is due to the fact that we decided to adopt the WS-Policy constructs for this purpose. In the future we will consider extending the policy and resource description modules with more expressive language constructs. This extension will involve the support to specify the Service-level Agreement (SLA) policy, the pricing policy, and the penalty and compensation policy of a cloud service.

- *Continuous validation and improvement of the BSL*: A concrete XSD template of the BSL has been created and distributed to our industry partners in the 4caast community [European Comission, 2010]. However, this XSD template still lacks a general acceptance and adoption of a larger cloud community and requires further validation activities in the future.

- *Extensions of the BMTs*: Based on the requirements derived from the 4caaSt project, we have discussed only the most important operators for engineering

CSBAs: the Insert, Delete, Query, Match, Link, Unlink, and Resolve operators. These operators aim to support the CSBA engineers and cloud service providers only at the design phase of the CSBA engineering lifecycle. Future research directions may target the runtime phase of the lifecycle and introduce more operators to support this phase.

- *The need of a Blueprint Constraint Language*: A CSBA configuration has been defined in Definition 1.3, Section 1.2 as a composition of blueprints to configure the deployment environment of an CSBA. Currently, there is a lack of a language used to express invariant constraints in an CSBA configuration. These constraints may come from different sources like the global Service Level Agreement (SLA) terms, deployment constraints, data residency constraints, auditability constraints, security constraints, etc. The CSBA engineers should be able to use this language to specify these constraints that will govern both the composition of blueprints at the design time and the behaviors of cloud services at runtime. We call such a language the *Blueprint Constraint Language*.

APPENDIX A: ACRONYMS AND GLOSSARY

4CaaSt Project acronym: The 4CaaSt project aims to create an advanced PaaS Cloud platform which supports the optimized and elastic hosting of internet-scale, multi-tier applications and embeds all the necessary features easing programming of rich applications and enabling the creation of a true business ecosystem where applications coming from different providers can be tailored to different users, mashed up and traded together [European Comission, 2010]

BMT Blueprint Manipulation Techniques: are defined as the techniques for publishing, querying, and composing blueprints with aim to support the design and configuration of an CSBA.

BSL Blueprint Specification Language: provides a means for cloud service providers to abstractly (i.e., independent of implementation) and unambiguously specify a cloud service in a blueprint.

CSBA Cloud Service-based Application: is a type of SBA that is developed by reusing and composing cloud services across all three layers of the cloud stack, i.e. SaaS, PaaS, and IaaS.

Configuration In communications or computer systems, a configuration is an arrangement of functional units according to their nature, number, and chief characteristics. Often, configuration pertains to the choice of hardware, software, firmware, and documentation. The configuration affects system function and performance. Source: Federal Standard 1037C. Link: `http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm`

Deployment According to the Open Grid Forum's CDDLM Foundation document the deployment can be defined as follows: "Deploying any complex, distributed service presents many challenges related to service configuration and management. These range from how to describe the precise, desired configuration of the service, to how we can automatically and repeatedly deploy, manage and

then remove the service. Deployment description challenges include how to represent the full range of service and resource elements, how to support service "templates" (where some description files can be used later on as a base for future deployment), service composition, correctness checking, and so on. Deployment challenges include automation, correct sequencing of operations across distributed resources, service lifecycle management, clean service removal, security, and so on. Addressing these challenges is highly relevant to Grid computing at a number of levels, including configuring and deploying individual Web Services (including WS-RF and other dialects), and composite systems made up of many co-operating Web Services.". Source: S-Cube Knowledge Model. Link: `http://www.s-cube-network.eu/km/terms/s/service-deployment`.

IaaS Infrastructure-as-a-Service: The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). [Mell & Grance, 2009]

OWL Web Ontology Language: is an ontology modeling language recommended by the World Wide Web Consortium. [McGuinness & van Harmelen (Eds.), 2004]

OVF Open Virtualization Format: is considered nowadays as an open standard for packaging and distributing virtual appliances. It contains a set of XML templates (conforming to predefined XSDs) to support the specification of either the offering of an IaaS provider or the infrastructure resource requirements of a SaaS or PaaS provider. [DMTF, b]

PaaS Platform-as-a-Service: The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. [Mell & Grance, 2009]

QoS Quality of Service: is a quality attribute sub-concept that represents those quality attributes that can be measured objectively or they are unmeasurable but can take objective values. For example, security attributes are QoS attributes but cannot be measured. However, the values that they take are objective. QoS attributes are typical constituents of SLAs (e.g response time and availability).Source:

S-Cube Knowledge Model. Link: http://www.s-cube-network.eu/km/terms/q/quality-of-service-qos.

RDF Resource Description Framework: is a standardized approach to describe resources on the Web and the relationships among the resources [Manola & Miller, 2004]

SaaS Software-as-a-Service: The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings. [Mell & Grance, 2009]

SBA Service-based Application: is composed by a number of possibly independent services, available in a network, which perform the desired functionalities of the architecture. Such services could be provided by third parties, not necessarily by the owner of the service-based application. Note that a service-based application shows a profound difference with respect to a component-based application: while the owner of the component-based application also owns and controls its components, the owner of a service-based application does not own, in general, the component services, nor it can control their execution. [Andrikopoulos et al., 2008]

SLA Service-level Agreement: is that part of a service contract where the level of service is formally defined. In practice, the term SLA is sometimes used to refer to the contracted delivery time (of the service) or performance. Source: S-Cube knowledge model. Link: http://www.s-cube-network.eu/km/terms/s/service-level-agreement

SOA Service-oriented Architecture: is a logical structure of loosely coupled and interoperable software services that can be easily shared within and between enterprises, via published and discoverable interfaces [Papazoglou, 2007]

UML Unified Modeling Language: is a standardized (ISO/IEC 19501:2005), general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. Source: Wikipedia. Link: http://en.wikipedia.org/wiki/Unified_Modeling_Language

Appendix B: Result of the Questionnaire Evaluation

In this appendix, we report the detailed results of the questionnaire evaluation of the Blueprint Approach that has been discussed in the previous Section 6.2.6.

Figure 7.1: Participant's Information(Results of Question 1,2 and 3)

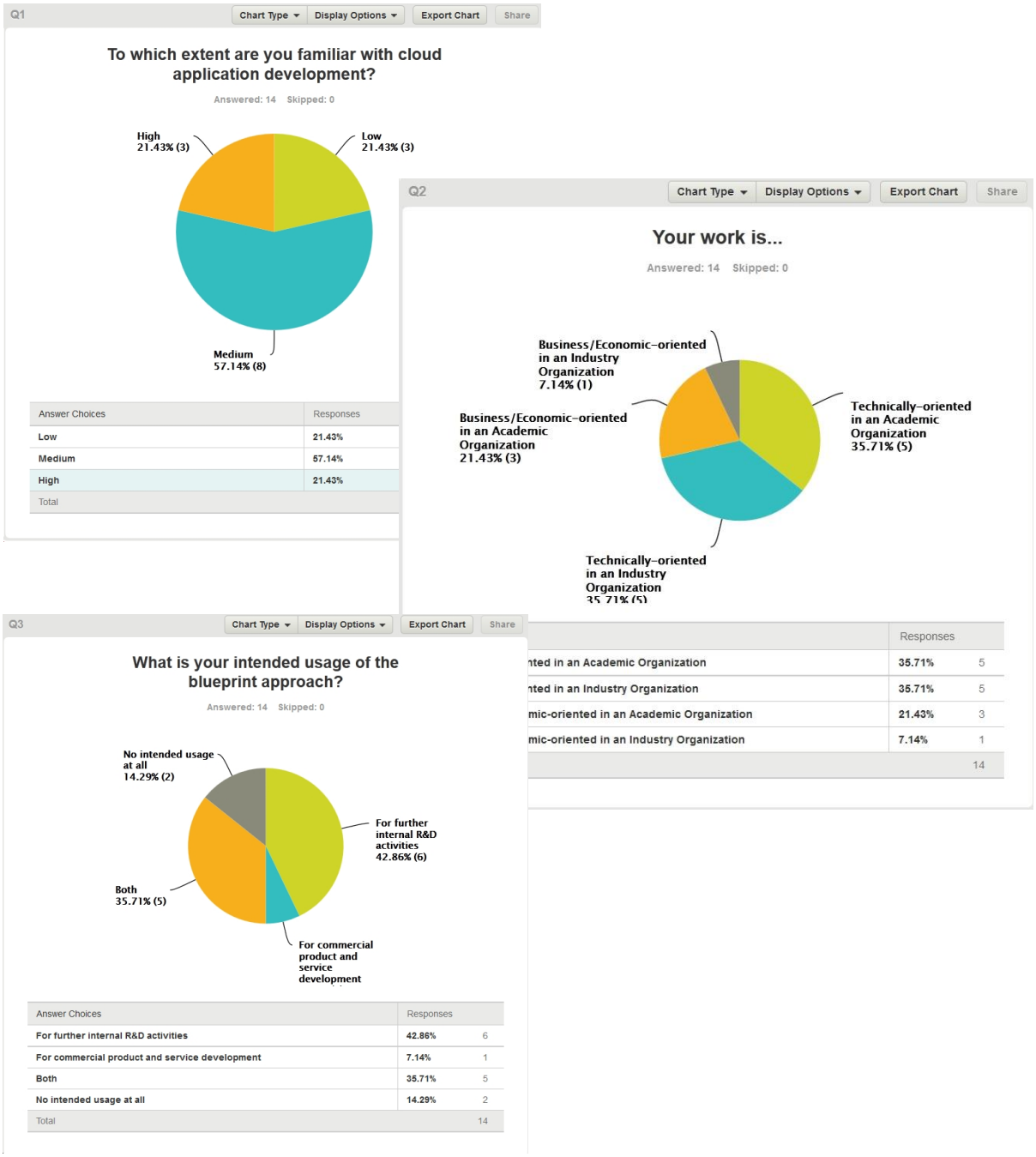


Figure 7.2: Evaluating the Blueprint Approach(Results of Question 4 and 5)

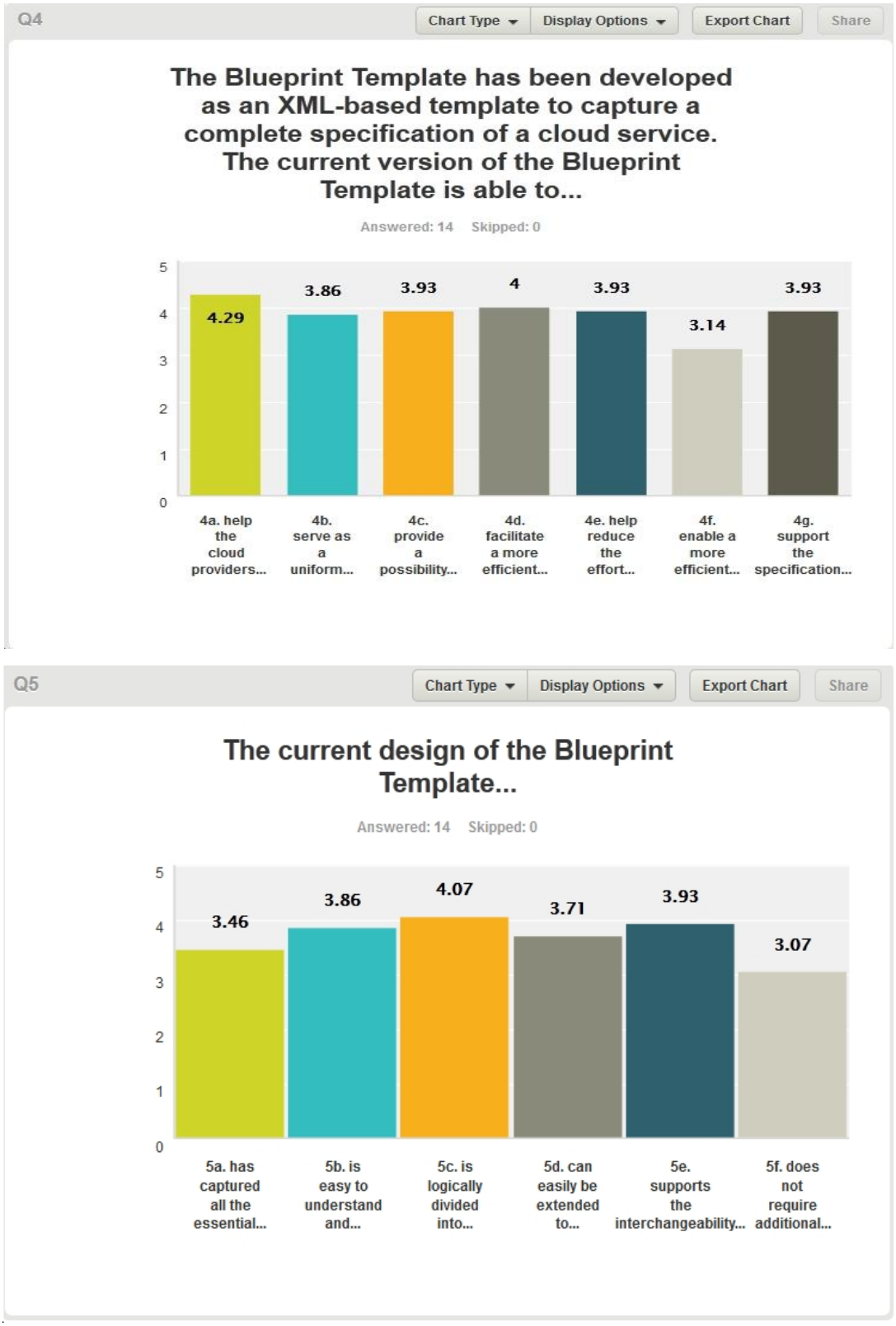
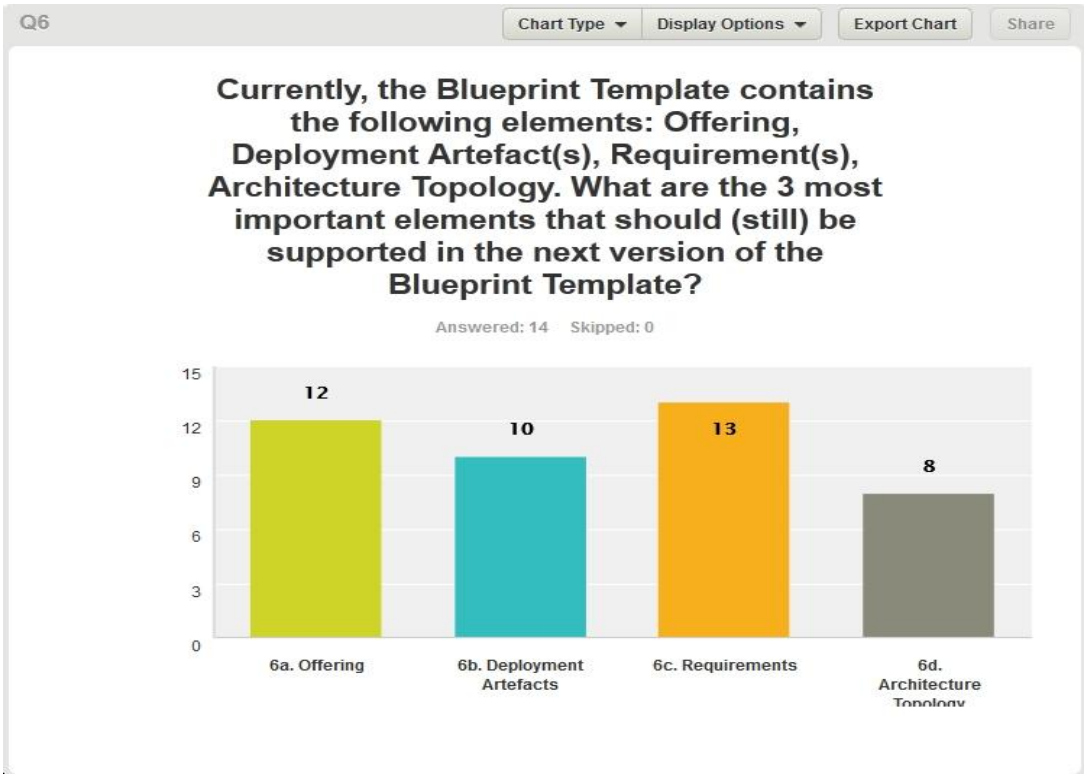


Figure 7.3: Evaluating the Blueprint Approach(Results of Question 6 and 7)



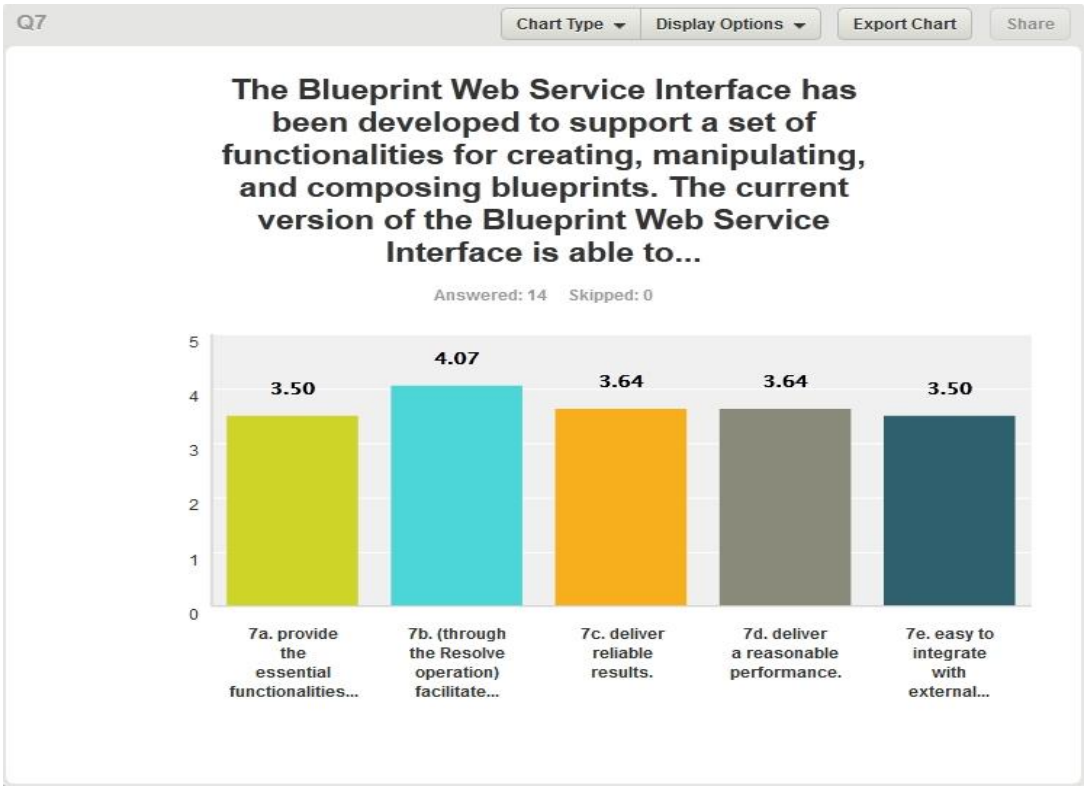
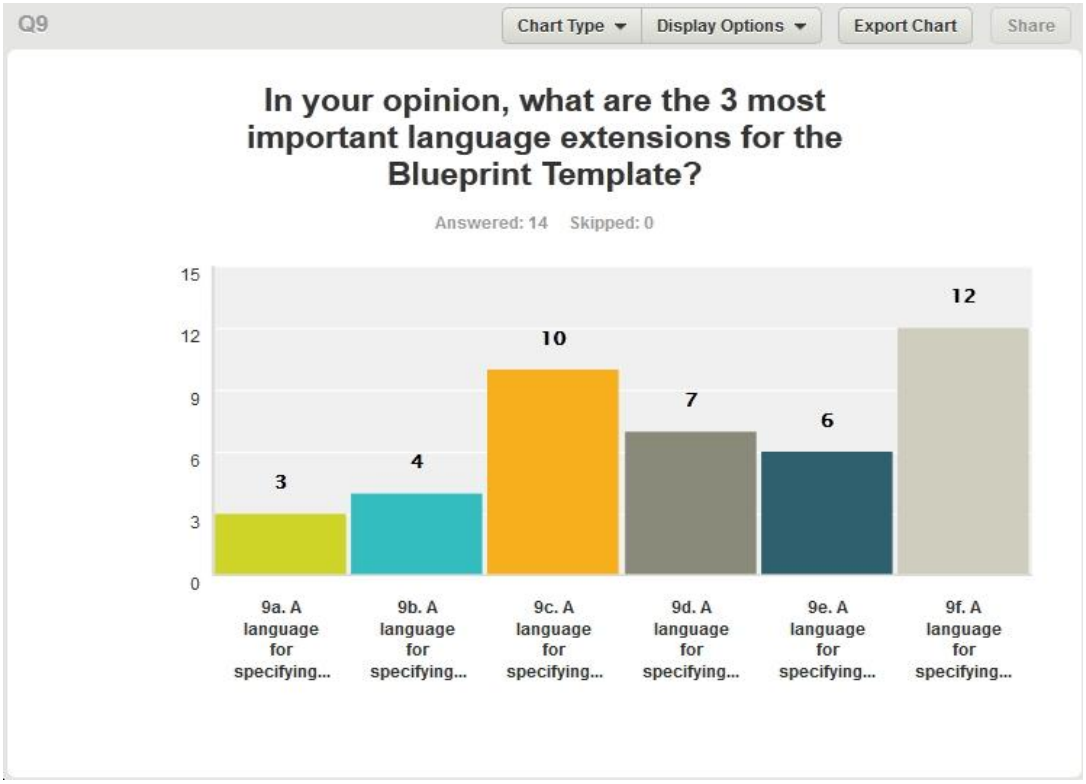
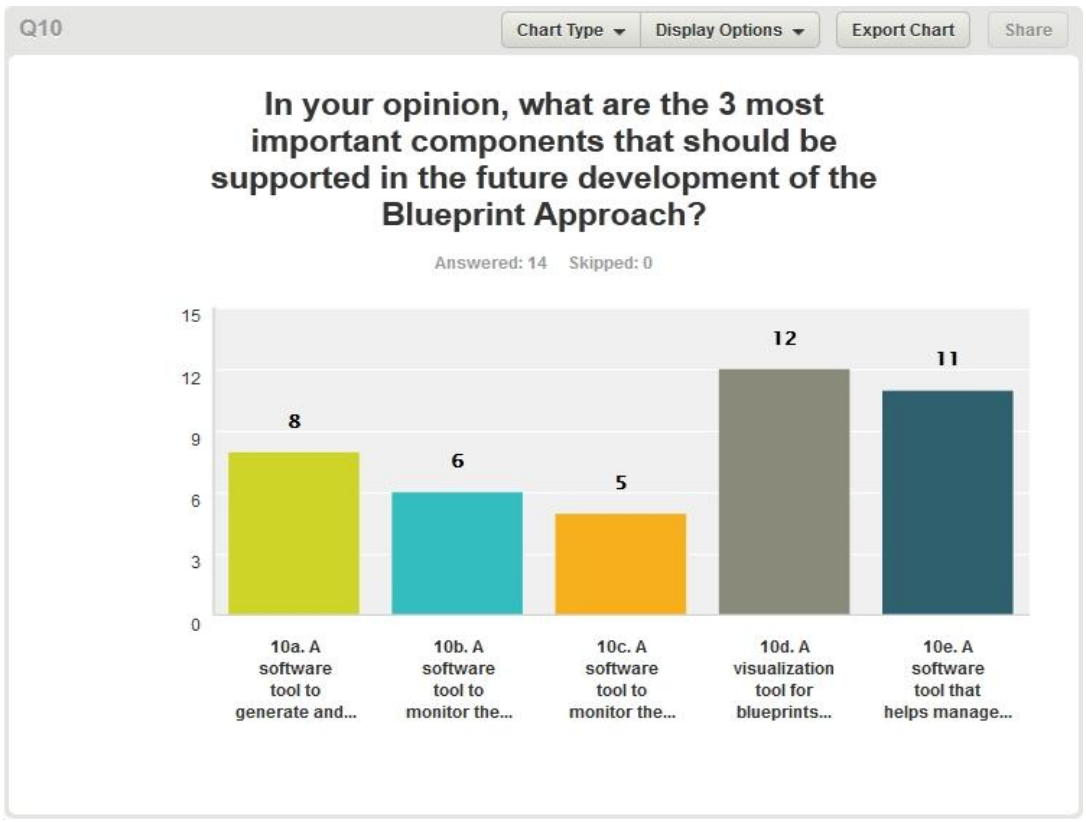


Figure 7.4: Improvement Suggestions(Results of Question 9 and 10)





BIBLIOGRAPHY

[Allmendinger & Lombreglia,] Allmendinger, G. & Lombreglia, R. Four Strategies for the Age of Smart Services. *Harvard Business Review*, 83(10), 131–145.

[Amazon,] Amazon. AWS Formation. online information.Available at http://aws.amazon.com/cloudformation/.

[Andrikopoulos, 2010] Andrikopoulos, V. (2010). *A Theory and Model for the Evolution of Software Services*. Open access publications from tilburg university.

[Andrikopoulos et al., 2010] Andrikopoulos, V., Fugini, M., Papazoglou, M. P., Parkin, M., Pernici, B., & Siadat, S. H. (2010). Qos contract formation and evolution. In *EC-Web* (pp. 119–130).

[Andrikopoulos et al., 2013] Andrikopoulos, V., Strauch, S., Exertier, F., Legrand, J., Momm, C., Vogel, J., Niemoeller, J., Lelli, F., Carrie, S., Moulos, V., Kranas, P., Moltchanov, B., Nguyen, D. K., Woodcock, K., Junker, F., Ivanovic, D., Wettinger, J., García, S., & Arozarena, P. (2013). Use Case Applications eMarketPlace for SMEs: Report on Integration. 4CaaSt Internal Project Deliverable 8.1.4.

[Andrikopoulos et al., 2008] Andrikopoulos et al., V. (2008). *State of the Art Report on Software Engineering Design Knowledge and Survey of HCI and Contextual Knowledge*. Project Deliverable PO-JRA-1.1.1, S-Cube Network of Excellence.

[Aoki, 2007] Aoki, O. (2007). Debian reference. available at http://www.debian.org/doc/manuals/debian-reference/index.en.html.

[Armbrust et al., 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., & Zaharia, M. (2009). *Above the Clouds: A Berkeley View of Cloud Computing*. Technical report.

[Arozarena et al., 2012] Arozarena, P., Dao, M., Exertier, F., Pelletier, B., Junker, F., Garcí, S., Spriestersbach, A., Andrikopoulos, V., Strauch, S., Moltchanov, B., Moulos, V., & Giesmann, A. (2012). Use Case Applications eMarketPlace for SMEs: Report on Experimentation Results. 4CaaSt Project Deliverable 8.1.5.

[Baader et al., 2008] Baader, F., Horrocks, I., & Sattler, U. (2008). Description Logics. In F. van Harmelen, V. Lifschitz, & B. Porter (Eds.), *Handbook of Knowledge Representation* chapter 3, (pp. 135–180). Elsevier.

[Balani, 2005] Balani, N. (2005). The Future of the Web is Semantic. *IBM DeveloperWorks*.

[Beckett, 2004] Beckett, D. (2004). Rdf/xml syntax specification. W3C Recommendation.

[Belguidoum & Dagnat, 2007] Belguidoum, M. & Dagnat, F. (2007). Dependency management in software component deployment. *Electron. Notes Theor. Comput. Sci.*, 182, 17–32.

[Bernstein et al., 2009] Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., & Morrow, M. (2009). Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In *Proceedings of the Fourth International Conference on Internet and Web Applications and Services*: IEEE Computer Society.

[Bernstein & Vij, 2010] Bernstein, D. & Vij, D. (2010). Intercloud directory and exchange protocol detail using xmpp and rdf. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10 (pp. 431–438). Washington, DC, USA: IEEE Computer Society.

[Bernstein et al., 2000] Bernstein, P. A., Halevy, A. Y., & Pottinger, R. A. (2000). A vision for management of complex models. *SIGMOD Rec.*, 29(4), 55–63.

[Beugnard et al., 1999] Beugnard, A., Jézéquel, J.-M., Plouzeau, N., & Watkins, D. (1999). Making components contract aware. *Computer*, 32(7), 38–45.

[Binz et al., 2012] Binz, T., Breiter, G., Leymann, F., & Spatzier, T. (2012). Portable Cloud Services Using TOSCA. *IEEE Internet Computing*, 16(03), 80–85.

[Binz et al., 2011] Binz, T., Exertier, F., Jiménez-Peris, R., Geoghegan, P., Oskarsson, J., Porcher, G., Pelletier, B., Riggs, S., Rodero-Merino, L., Souillard, C., Spriestersbach, A., & Strauch, S. (2011). Immigrant paas technologies: Components design and open specification d7.2.1. 4CaaSt project Deliverable.

[Brickley & Guha, 2004] Brickley, D. & Guha, R. (2004). Rdf vocabulary description language 1.0: Rdf schema. W3C Recommendation.

[Brunelière et al., 2010] Brunelière, H., Cabot, J., & Frédéric, J. (2010). Combining model-driven engineering and cloud computing. In *Proceedings of the 4th edition of Modeling, Design, and Analysis for the Service Cloud*.

[Buyya et al., 2010] Buyya, R., Ranjan, R., & Calheiros, R. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the ICA3PP 2010*, volume LNCS 6081: Springer.

[Cai et al., 2009] Cai, H., Zhang, K., Wang, M., Li, J., Sun, L., & Mao, X. (2009). Customer centric cloud service model and a case study on commerce as a service. In *Proceedings of the IEEE International Conference on Cloud Computing*.

[Chapman et al., 2010] Chapman, C., Emmerich, W., Márquez, F. G., Clayman, S., & Galis, A. (2010). Software Architecture Definition for On-Demand Cloud Provisioning. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)* (pp. 61–72).

[Chen et al., 2005] Chen, K., Sztipanovits, J., & Neema, S. (2005). Toward a semantic anchoring infrastructure for domain-specific modeling languages. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05 (pp. 35–43). New York, NY, USA: ACM.

[Chieu et al., 2010] Chieu, T., Mohindra, A., Karve, A., & Segal, A. (2010). Solution-Based Deployment of Complex Application Services on a Cloud. In *Proceedings of the IEEE International Conference on Service Operations and Logistics and Informatics (SOLI)*.

[Collazo-mojica et al., 2010] Collazo-mojica, X. J., Sadjadi, S. M., Kon, F., & Silva, D. D. (2010). Virtual environments: Easy modeling of interdependent virtual appliances in the cloud.

[Dean & Schreiber, 2004] Dean, M. & Schreiber, G. (2004). Owl web ontology language reference. W3C Recommendation.

[DMTF, a] DMTF. DMTF to Develop Standards for Managing a Cloud Computing Environment, http://www.dmtf.org/standards/cloud.

[DMTF, b] DMTF. Open Virtualization Format (OVF), http://www.dmtf.org/standards/ovf.

[Do & Rahm, 2002] Do, H.-H. & Rahm, E. (2002). Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02 (pp. 610–621).: VLDB Endowment.

[Elvesæter et al., 2011] Elvesæter, B., Berre, A.-J., & Sadovykh, A. (2011). Specifying services using the service oriented architecture modeling language (soaml) - a baseline for specification of cloud-based services. In *CLOSER* (pp. 276–285).

[Emerson et al., 2004] Emerson, M. J., Sztipanovits, J., & Bapty, T. (2004). A mof-based metamodeling environment. *j-jucs*, 10(10), 1357–1382.

[European Comission, 2010] European Comission (2010). 4CaaSt: Building the PaaS Cloud of the Future. Information & Communication Technologies Unit. Project Objectives Document.

[European Commision, 2009] European Commision (2009). European union international trade in services - analytical aspects - data 2003 - 2007.

[Everware-CBDI,] Everware-CBDI. CBDI-SAE Meta Model for SOA v2. CBDI-SAE specification.

[Fitzsimmons & Fitzsimmons, 2004] Fitzsimmons, J. A. & Fitzsimmons, M. J. (March 2004). *Service Management: Operations, Strategy, and Information Technology*. Irwin Professional Pub, 4th edition.

[Forrester Research Inc, 2011] Forrester Research Inc (2011). Research : Sizing the cloud.

[Franch et al., 1999] Franch, X., Pinyol, J., & Vancells, J. (1999). Browsing a component library using non-functional information. In *Proceedings of the 1999 Ada-Europe International Conference on Reliable Software Technologies*, Ada-Europe '99 (pp. 332–343). London, UK, UK: Springer-Verlag.

[Galán et al., 2009] Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., & Vaquero, L. M. (2009). Service Specification in Cloud Environments Based on Extensions to Open Standards. In *Proceedings of the Fourth International ICST Conference on Communication System Software & Middleware (COMSWARE '09)* (pp. 1–12).

[Gasevic et al., 2004] Gasevic, D., Djuric, D., Devedzic, V., & Damjanovi, V. (2004). Converting uml to owl ontologies. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04 (pp. 488–489). New York, NY, USA: ACM.

[Gehlert & Metzger, 2008] Gehlert, A. & Metzger, A. (2008). Quality reference model for sba. S-Cube project deliverable CD-JRA-1.3.2.

[Gómez et al., 2012] Gómez, S. G., Carrié, S., Spriestersbach, A., Binz, T., Moltchanov, B., Strauch, S., Toth, D., & Arozarena, P. (2012). Use case applications emarketplace for smes: Scenario definition d8.1.2. 4CaaSt Project Deliverable 8.1.2.

[Gómez et al., 2011] Gómez, S. G., Vogel, J., Giessmann, A., Menychtas, A., & Gatzioura, A. (2011). Marketplace functions: Components design and open specification. Project deliverable D3.2.1 (Month 12), EU FP7 project 4caaSt.

[Gómez et al., 2012] Gómez et al. (2012). A 4caast whitepaper, 4caast technical value proposition, version 1.0.

[Goldsack et al., 2009] Goldsack, P., Guijarro, J., Loughran, S., Coles, A., Farrell, A., Lain, A., Murray, P., & Toft, P. (2009). The smartfrog configuration management framework. *SIGOPS Oper. Syst. Rev.*, 43(1), 16–25.

[Guttag, 1977] Guttag, J. (1977). Abstract data types and the development of data structures. *Commun. ACM*, 20(6), 396–404.

[Hamdaqa et al., 2011] Hamdaqa, M., Livogiannis, T., & Tahvildari, L. (2011). A reference model for developing cloud applications. In *In proceedings of CLOSER'11*.

[Hart et al., 2004] Hart, L., Emery, P., Colomb, B., Raymond, K., Taraporewalla, S., Chang, D., Ye, Y., Kendall, E., & Dutra, M. (2004). *OWL Full and UML 2.0 Compared*. Technical report.

[Hevner et al., 2004] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Q.*, 28(1), 75–105.

[Horrocks et al., 2007] Horrocks, I., Patel-Schneider, P. F., McGuinness, D. L., & Welty, C. A. (2007). OWL: a Description Logic Based Ontology Language for the Semantic Web. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition)* chapter 14. Cambridge University Press.

[Intel, 2010] Intel (2010). Intel cloud computing taxonomy and ecosystem analysis. IT@Intel Brief, Intel Information Technology.

[Internet of Services,] Internet of Services. Unified service description language (usdl) specification. available online at http://www.internet-of-services.com/index.php?id=288&L=0.

[Kazarian & Hanlon, 2010] Kazarian, B. & Hanlon, B. (2010). Smb cloud adoption study dec 2010 - global report. Microsoft Press.

[Keahey et al., 2009] Keahey, K., Tsugawa, M., Matsunaga, A., & Fortes, J. (2009). Sky computing. *IEEE Internet Computing*, 13(5), 43–51.

[Kleppe et al., 2003] Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture : Practice & Promise*. Addison-Wesley Professional.

[Klyne & Caroll, 2004] Klyne, G. & Caroll, J. J. (2004). Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation.

[Konstantinou et al., 2009] Konstantinou, A. V., Eilam, T., Kalantar, M., Totok, A. A., Arnold, W., & Snible, E. (2009). An Architecture for Virtual Solution Composition & Deployment in Infrastructure Clouds. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing (VTDC '09)* (pp. 9–18).

[Krill, 2009] Krill, P. (2009). The cloud-soa connection. InfoWorld article.

[Liu & Zic, 2011] Liu, D. & Zic, J. (2011). Cloud#: A specification language for modeling cloud. In *IEEE CLOUD* (pp. 533–540).

[Lusch et al., 2008] Lusch, R. F., Vargo, S. L., & Wessels, G. (2008). Toward a conceptual foundation for service science: contributions from service-dominant logic. *IBM Syst. J.*, 47(1), 5–13.

[Madhavan et al., 2001] Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01 (pp. 49–58). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Manola & Miller, 2004] Manola, F. & Miller, E. (2004). RDF Primer. W3C Recommendation.

[Maximilien et al., 2009] Maximilien, E. M., Ranabahu, A., Engehausen, R., & Anderson, L. C. (2009). Toward Cloud-Agnostic Middlewares. In *Proceeding of the 24th ACM SIGPLAN conference on Object Oriented Programming Systems Languages & Applications* (pp. 619–626).

[McGuinness & van Harmelen (Eds.), 2004] McGuinness, D. L. & van Harmelen (Eds.), F. (2004). *OWL Web Ontology Language Overview*. W3C Recommendation.

[Mell & Grance, 2009] Mell, P. & Grance, T. (2009). The NIST Definition of Cloud Computing. National Institute of Standards and Technology, Information Technology Laboratory.

[Melnik, 2004] Melnik, S. (2004). *Generic Model Management: Concepts And Algorithms (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

[Melnik et al., 2003] Melnik, S., Rahm, E., & Bernstein, P. A. (2003). Rondo: a programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03 (pp. 193–204). New York, NY, USA: ACM.

[Microsoft MSDN, 2011] Microsoft MSDN (2011). Windows Azure Schema Reference. online.

[Mietzner, 2010] Mietzner, R. (2010). *A Method and Implementation to Define and Provision Variable Composite Applications, and its usage in Cloud Computing*. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany.

[Mietzner et al., 2009] Mietzner, R., van Lessen, T., Wiese, A., Wieland, M., Karastoyanova, D., & Leymann, F. (2009). Virtualizing Services and Resources with ProBus: The WS-Policy-Aware Service and Resource Bus. In *Proceedings of the IEEE International Conference on Web Services (ICWS* (pp. 617–624).

[Mili et al., 1995] Mili, H., Mili, F., & Mili, A. (1995). Reusing software: Issues and research directions. *IEEE Trans. Softw. Eng.*, 21(6), 528–562.

[Motik et al., 2009] Motik, B., Parsia, B., & Patel-Schneider, P. F. (2009). Owl 2 web ontology language - xml serialization. W3C Recommendation.

[Nguyen et al., 2012a] Nguyen, D. K., Lelli, F., Papazoglou, M. P., & van den Heuvel, W.-J. (2012a). Blueprinting approach in support of cloud computing. *Future Internet*, 4(1), 322–346.

[Nguyen et al., 2012b] Nguyen, D. K., Lelli, F., Papazoglou, M. P., & van den Heuvel, W.-J. (2012b). Issue in automatic combination of cloud services. In *ISPA* (pp. 487–493).

[Nguyen et al., 2011] Nguyen, D. K., Lelli, F., Taher, Y., Parkin, M., Papazoglou, M. P., & van den Heuvel, W.-J. (2011). Blueprint Template Support for Cloud-based Service Engineering. In *Proceedings of the 4th European ServiceWave Conference*.

[Nguyen et al., 2012c] Nguyen, D. K., Taher, Y., Papazoglou, M. P., & van den Heuvel, W.-J. (2012c). Service-based application development on the cloud - state of the art and shortcomings analysis. In *CLOSER* (pp. 395–400).

[Niemoeller et al., 2009] Niemoeller, J., Fikouras, I., de Rooij, F., Klostermann, L., Stringer, U., & Olsson, U. (2009). Ericsson Composition Engine - Next-generation IN. *Ericsson Review*, 02, 22–27.

[OASIS, 2006] OASIS (2006). Reference Model for Service Oriented Architecture 1.0 . OASIS Standard.

[OASIS, 2007] OASIS (2007). Web Services Business Process Execution Language Version 2.0. OASIS standard.

[OASIS, 2013] OASIS (2013). Oasis topology and orchestration specification for cloud applications (tosca) v1.0. OASIS Specification (ongoing).

[OASIS, 2008] OASIS (September 2008). *Solution deployment descriptor specification 1.0, http://docs.oasis-open.org/sdd/v1.0/os/sdd-spec-v1.0-os.pdf*. Technical report, OASIS.

[O'Brien & Marakas, 2009] O'Brien, J. & Marakas, G. (2009). *Introduction to Information Systems*. McGraw-Hill Education.

[OCCI-Working Group, 2011] OCCI-Working Group (2011). Open cloud computing interface - infrastructure.

[OMG, 2009] OMG (2009). Ontology definition metamodel (odm) version 1.0. OMG Specification.

[OMG, 2011] OMG (2011). Meta object facility (mof) 2.0 query/view/transformation, v1.1. OMG Standard.

[OMG, 2011] OMG (2011). Unified modeling language (uml), v2.4.1. OMG Specification.

[OMG, 2012] OMG (2012). Service oriented architecture modeling language (soaml) specification, v1.0.1. OMG Specification.

[O'Sullivan, 2006] O'Sullivan, J. J. (2006). *Towards a precise understanding of service properties*. PhD thesis, Queensland University of Technology.

[Pahl et al., 2009] Pahl, C., Giesecke, S., & Hasselbring, W. (2009). Ontology-based modelling of architectural styles. *Inf. Softw. Technol.*, 51(12), 1739–1749.

[Papazoglou, 2003] Papazoglou, M. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Fourth International Conference on Web Information Systems Engineering (WISE'03)* (pp. 3–).

[Papazoglou & Vaquero, 2012] Papazoglou, M. & Vaquero, L. (2012). *Knowledge Service Engineering Handbook*, chapter Knowledge-Intensive Cloud Services: Transforming the Cloud Delivery Stack, (pp. 449–494). CRC Press.

[Papazoglou, 2007] Papazoglou, M. P. (2007). *Web Services: Principles and Technology*. Prentice Hall.

[Papazoglou, 2012] Papazoglou, M. P. (2012). *Web Services: Principles and Technology*. Pearson Education Canada, 2th edition.

[Papazoglou & van den Heuvel, 2011] Papazoglou, M. P. & van den Heuvel, W.-J. (2011). Blueprinting the cloud. *IEEE Internet Computing*, 15(6), 74–79.

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., & Horrocks, I. (2004). Owl web ontology language: Semantics and abstract syntax. W3C Recommendation.

[Rochwerger et al., 2009] Rochwerger et al., B. (2009). The Reservoir Model & Architecture for Open Federated Cloud Computing. *IBM Journal of Research & Development*, 53(4).

[Rodero-Merino et al., 2010] Rodero-Merino, L., Vaquero, L. M., Gil, V., Galán, F., Fontán, J., Montero, R. S., & Llorente, I. M. (2010). From infrastructure delivery to service management in clouds. *Future Gener. Comput. Syst.*, 26(8), 1226–1240.

[SAP A.G., 2011] SAP A.G. (2011). The sap service marketplace.

[Suciu, 1998] Suciu, D. (1998). Semistructured data and xml.

[Sun et al., 2012a] Sun, L., Dong, H., & Ashraf, J. (2012a). Survey of service description languages and their issues in cloud computing. In *Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on* (pp. 128–135).: IEEE.

[Sun et al., 2012b] Sun, Y. L., Harmer, T., & Stewart, A. (2012b). Specifying cloud application requirements: an ontological approach. In *CloudComp 2012*.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd edition.

[Taher et al., 2011] Taher, Y., Lelli, F., Nguyen, D. K., & Parkin, M. (2011). Service engineering and lifecycle management - blueprinting the cloud: Scientific and technical report. Project deliverable D2.1.1 (Month 12), EU FP7 project 4caaSt.

[Taher et al., 2012] Taher, Y., Nguyen, D. K., Lelli, F., van den Heuvel, W.-J., & Papazoglou, M. P. (2012). On engineering cloud applications-state of the art, shortcomings analysis, and approach. *Scalable Computing: Practice and Experience*, 13(3).

[Thrash, 2010] Thrash, R. (2010). Building a cloud computing specification: fundamental engineering for optimizing cloud computing initialtives. Computer Science Corporation (CSC) Whitepaper.

[Tsai et al., 2010] Tsai, W.-T., Sun, X., & Balasooriya, J. (2010). Service-oriented cloud computing architecture. *2010 Seventh International Conference on Information Technology New Generations*, (pp. 684–689).

[Vambenepe, 2009] Vambenepe, W. (2009). Reality check on cloud portability. online.

[van den Heuvel, 2009] van den Heuvel, W.-J. (2009). *Changing the Face of the Global Digital Economy - Smart Service Networks as a Catalyst for Innovation: Inaugural Speech by W. J. A. M. van den Heuvel*. University Tilburg.

[VMWare,] VMWare. VMware vCenter Orchestrator. available online: http://www.vmware.com/products/vcenter-orchestrator/overview.html.

[W3C,] W3C. Owl-s: Semantic markup for web services. Web Ontology Working Group, available at http://www.ai.sri.com/daml/services/owl-s/1.2/overview/.

[W3C, 1999a] W3C (1999a). XML Path Language (XPath) Version 1.0. W3C Recommendation.

[W3C, 1999b] W3C (1999b). Xsl transformations (xslt) version 1.0. W3C Recommendation.

[W3C, 2004] W3C (2004). Owl web ontology language overview. W3C Recommendation.

[W3C, 2005a] W3C (2005a). Web Service Modeling Ontology (WSMO). W3C Member Submission.

[W3C, 2005b] W3C (2005b). Web Service Semantics - WSDL-S. W3C Member Submission.

[W3C, 2006] W3C (2006). Web Services Policy 1.2 - Framework (WS-Policy). W3C Member Submission.

[W3C, 2008] W3C (2008). Sparql query language for rdf. W3C Recommendation.

[W3C, 2011] W3C (2011). Web Services Description Language (WSDL) 1.1. W3C Note.

[W3C, 2012] W3C (2012). Sparql 1.1 update. W3C Proposed Recommendation.

[W3C, 2013] W3C (2013). XQuery 3.0: An XML Query Language. W3C Candidate Recommendation.

All links and references are last checked on the 06.09.2013

SIKS Dissertation Series

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/ Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennis-technologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)

Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design:

Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)

Using Multiple Models of Reality:
On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations
in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for
Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of
Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models
for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating
multidisciplinary design teams

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing
Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the
Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving
Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars
for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation
for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building
Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for
Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative
Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering
- A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen
van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on
the Semantic Web; Exploring how
semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on
Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for
probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art
and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery
in Database Systems by Exploiting
Application Semantics

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use
of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in
learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlović (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development and management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramírez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

2008-01 Katalin Boer-Sorbán (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos (CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Withdrawn

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of

Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness:
An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling
for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based
Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of
Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure:
Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering -
A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of
Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of
Archival Description and Access

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances
in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination
with Virtual Humans On Specifying, Scheduling and
Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU))
Analysis and Validation of
Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication -
Emotion Regulation and Involvement-Distance
Trade-Offs in Embodied Conversational
Agents and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through
autonomous management of design patterns

2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting
Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP):
Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches
for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric
Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence:
Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design:
a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data,
Applications for Preference Learning
and Supervised Network Inference

2011-38
Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents
in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in

Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair (VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)

Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24
Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN)
A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN)
Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN)
A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU)
Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT)
Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT)
Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU)
Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU)
Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU)
On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT)
A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA)
Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA)
Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE)
Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM)
Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD)
Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD)
Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and

Scalable Stream Processing

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati(TUD)
Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT)
Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves(CWI)
The Data Cyclotron: Juggling Data and
Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT)
Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk(VU)
Making enemies: cognitive modeling for
opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN)
Metagenomic Data Analysis:
Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige(UvT)
A Unified Modeling Framework for Service Design

2013-11 Evangelos Pournaras(TUD)
Multi-level Reconfigurable
Self-organization in Overlay Services

2013-12 Marian Razavian(VU)
Knowledge-driven Migration to Services

2013-13 Mohammad Safiri(UT)
Service Tailoring: User-centric creation of
integrated IT-based homecare services
to support independent living of elderly

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to
Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)
Multiagent Learning -
Dynamic Games and Applications

2013-16 Eric Kok (UU)
Exploring the practical benefits of
argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination
for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD)
Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA)
Fast and Reliable Online Learning
to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)
Text-to-text generation by
monolingual machine translation

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic

2013-23 Patricio de Alencar Silva(UvT)
Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)
Intention-based Decision Support.
A new way of representing and implementing clinical
guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic
Homecare Service Provisioning

2013-27 Mohammad Huq (UT)
Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT)
When Complexity becomes Interesting:
An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT)
Listening Heads

2013-30 Joyce Nakatumba (TUE)
Resource-Aware Business Process Management:
Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT)
Blueprint Model and Language for
Engineering Cloud Applications